Automated Deep Learning : Principles and Practice

Zhengying Liu

▶ **To cite this version:**

Zhengying Liu. Automated Deep Learning : Principles and Practice. Machine Learning [cs.LG]. Université Paris-Saclay, 2021. English. NNT : 2021UPASG094 . tel-03464519v1

HAL Id: tel-03464519

https://theses.hal.science/tel-03464519v1

Submitted on 3 Dec 2021 (v1), last revised 6 Dec 2021 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université
PARIS-SACLAY

Automated Deep Learning: Principles and Practice

Apprentissage profond automatisé : principes et pratique

Thèse de doctorat de l'université Paris-Saclay

Ecole doctorale n° 580 : sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Mathématiques et informatique
Graduate School : Informatique et sciences du numérique
Référent : Faculté des sciences d'Orsay

Thèse préparée dans le **Laboratoire interdisciplinaire des sciences du numérique** (Université Paris-Saclay, CNRS), sous la direction d'**Isabelle GUYON**, Professeure, le co-encadrement de **Michèle SEBAG**, Dir. Recherche

Thèse soutenue à Paris-Saclay, le 9 novembre 2021, par

Zhengying LIU

Composition du jury

Emmanuel VAZQUEZ Professeur, CentraleSupelec, France	Président
Florence D'ALCHÉ-BUC Professeur, Télécom Paris, France	Rapporteur & Examinatrice
Yves GRANDVALET Dir. Recherche, Université de Technologie de Compiègne, France	Rapporteur & Examinateur
Joaquin VANSCHOREN Professeur Assistant, Eindhoven University of Technology, Pays-Bas	Examinateur
Jan N. VAN RIJN Professeur Assistant, Leiden University, Pays-Bas	Examinateur
Hugo Jair ESCALANTE Professeur, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Mexique	Examinateur
Isabelle GUYON Professeure, INRIA/Université Paris-Saclay, France	Directrice de thèse

THESE DE DOCTORAT
NNT : 2021UPASG094

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Zhengying LIU
November 2021

Acknowledgements

This thesis benefited from the kind support and help from many people.

I would like to express my sincere and deep gratitude to my advisors Isabelle Guyon and Michèle Sebag. They led me into the field of AutoML and guided me with passion and patience. They kindly provided me many opportunities to establish academic connections and to enhance my visibility in the academia. They helped me with all their heart even if they were already very busy. And I cannot emphasize more how great their help has been in difficult times.

Many collaborations have enriched this thesis. I would like to thank Olivier Bousquet, André Elisseeff and Mahsa Behzadi from Google Zurich for launching the AutoDL project and providing continuous support. I would like to thank Sergio Escalera (U. Barcelona, Spain and ChaLearn, USA), Julio Jacques Junior (Universitat Oberta de Catalunya) and Meysam Madadi (Universitat Autònoma de Barcelona, Spain) for kindly receiving me at Barcelona for an enlightening academic visit and also for their constant contributions in organizing and analyzing AutoDL challenges. I would like to thank my INRIA/LRI colleagues Marc Schoenauer, Herilalaina Rakotoarison, Lisheng Sun, Benjamin Donnot, Balthazar Donon for uncountable interesting discussions and ideas. Collaborations with them are a real pleasure. Specifically, the work on GramNAS with MCTS as search strategy (GramNAS-MCTS) is the result of a close collaboration with Herilalaina Rakotoarison. The work on GramNAS with evolution as strategy (GramNAS-AgEBO) is a collaboration with Romain Egelé during his internship in our laboratory. The work on LEAP nets applied to power systems was led by Benjamin Donnot and Balthazar Donon, with the help of Marc Schoenauer, Antoine Marot and Patrick Panciatici. I would like to thank my interns Zhen Xu, Romain Egelé and Adrian El Baz for their passion, hard-work and many interesting new ideas they brought in. Exchanges with outstanding colleagues such as Frank Hutter (University of Freiburg), Joaquin Vanschoren (Eindhoven University of Technology), Jan van Rijn (Leiden University), Sebastien Treguer (la Paillasse) helped a lot in shaping my ideas and guiding my future research. Concretely, Zhen Xu gave a lot of useful suggestions in forming the three-level formulation and helped preparing and launching AutoCV challenge and AutoCV2 challenge (the first two challenges in the AutoDL challenges) during his internship. The work on MetaDL challenge was done with Adrian El Baz during his internship in the TAU team at our lab, with the support of Jan van Rijn,

Sebastien Treguer, Joaquin Vanschoren, Adrien Pavao and many others. The GPU support was kindly provided by Microsoft.

For the AutoDL challenges which constitute a major part of this thesis, I am in deep gratitude to many people and organizations. These include the major organizors such as Olivier Bousquet (Google, Switzerland), André Elisseef (Google, Switzerland) Wei-Wei Tu (4paradigm, China) Adrien Pavao (U. Paris-Saclay; INRIA, France and ChaLearn, USA), Zhen Xu (4paradigm, China), Sergio Escalera (U. Barcelona, Spain and ChaLearn, USA), Julio Jacques Junior (Universitat Oberta de Catalunya, Spain), Meysam Madadi (Universitat Autònoma de Barcelona, Spain), and Sebastien Treger (La Pallaisse, France); and to many people who donated their data, gave good suggestions or helped beta-tests. A non-exhaustive list of them is as follows : Stephane Ayache (AMU, France), Hubert Jacob Banville (INRIA, France), Mahsa Behzadi (Google, Switzerland), Kristin Bennett (RPI, New York, USA), Hugo Jair Escalante (IANOE, Mexico and ChaLearn, USA), Gavin Cawley (U. East Anglia, UK), Baiyu Chen (UC Berkeley, USA), Albert Clapes i Sintes (U. Barcelona, Spain), Bram van Ginneken (Radboud U. Nijmegen, The Netherlands), Alexandre Gramfort (U. Paris-Saclay; INRIA, France), Yi-Qi Hu (4paradigm, China), Julio Jacques Jr. (U. Barcelona, Spain), Tatiana Merkulova (Google, Switzerland), Shangeth Rajaa (BITS Pilani, India), Herilalaina Rakotoarison (U. Paris-Saclay, INRIA, France), Mehreen Saeed (FAST Nat. U. Lahore, Pakistan), Marc Schoenauer (U. Paris-Saclay, INRIA, France), Michele Sebag (U. Paris-Saclay; CNRS, France), Danny Silver (Acadia University, Canada), Lisheng Sun (U. Paris-Saclay; UPSud, France), Fengfu Li (4paradigm, China), Lichuan Xiang (4paradigm, China), Jun Wan (Chinese Academy of Sciences, China), Mengshuo Wang (4paradigm, China), Jingsong Wang (4paradigm, China), Ju Xu (4paradigm, China). All challenges organized by us ran on the Codalab platform, administered by Université Paris-Saclay and maintained by CKCollab LLC, with primary developers Eric Carmichael (CKCollab, USA) and Tyler Thomas (CKCollab, USA). ChaLearn was the challenge organization coordinator. Google was the primary sponsor of the AutoDL challenges and helped defining the tasks, protocol, and data formats. 4Paradigm donated prizes, datasets, and contributed to the protocol, baselines methods and beta-testing. Other institutions of the co-organizers provided in-kind contributions, including datasets, data formatting, baseline methods, and beta-testing. In addition to the organizational aspect, I would also like to thank all the participating teams and individuals of the AutoDL challenges. It was their passion and efforts that made the competitions interesting and instructive. Many of them also contributed to the post-challenge analysis paper (Liu et al., 2021) accepted at TPAMI.

These including teams *automl_freiburg*, *DeepBlueAI*, *DeepWisdom*, *upwind_flys*, *PASA_NJU*.

I would like to warmly thank jury members in my PhD defense including the two reviewers Florence d'Alché-Buc (Telecom Paris), Yves Grandvalet (Université de Technologie de Compiègne) and Emmanuel Vazquez (CentraleSupelec), Joaquin Vanschoren (Eindhoven University of Technology), Jan van Rijn (Leiden University), Hugo Jair Escalante (National Institute of Astrophysics, Optics and Electronics (INAOE)).

I would like to thank Ecole polytechnique for granting me the 3-year scholarship for this PhD. And I would like to thank INRIA for providing support for attending conferences and seminars. I would like to thank Jean Zay project from IDRIS for providing computational resources for the GramNAS project.

Last but not least, I would like to thank my family. My wife has provided countless support for finishing my PhD, to which I am extremely grateful. She has been very caring and patient. Finally I express my gratitude to my parents, who made who I am, led me to an academic career and sent their constant regards.

Abstract

Automated Machine Learning (AutoML) aims at rendering the application of machine learning methods as devoid of human intervention as possible. This ambitious goal has been the object of much research and engineering ever since the outset of ML. While reaching fully automated ML on any possible application may still remain out-of-reach for several decades, there are practical and fundamental motivations to advance the state-of-the-art in this field. The objective of this thesis is to put a formal framework around this multi-faceted problem, to benchmark existing methods, and to explore new directions.

In previous work, AutoML is often synonymous of automated *hyperparameter optimization* (HPO) and/or *model selection* for a particular learning problem, as defined by some training data. The final performance evaluation is carried out on a test set drawn for the same data distribution as the training data (*i.i.d.* assumption). Neither training nor test duration are usually considered in this problem setting.

Our definition of the AutoML problem makes several departures from this simplified setting: For one, we go beyond the "single task" case and plunge the AutoML problem is the broader context of a *family of tasks* of similar nature. We therefore include in our setting the problems of *transfer learning* whose goal is to transfer "knowledge" from task to task, be it for *multi-task learning* or *domain adaptation*. Transfer learning can be tackled with various forms of *meta-learning* by making use of examples of tasks from the family of interest to "meta-train" the learning machine. Secondly, we define learning tasks in a more realistic and pragmatic way: a task includes not only a dataset (split into a training and test set), but also a metric of evaluation, a time budget (for training and testing), and well-defined computational resources (including memory constraints).

To formulate the AutoML problem in a rigorous way, we first introduce a mathematical framework that: (1) categorizes all involved algorithms into three levels (α, β and γ levels); (2) concretely defines the concept of a *task* (especially in a supervised learning setting); (3) formally defines HPO and meta-learning; (4) introduces an *any-time learning metric* that allows to evaluate learning algorithms by not only their accuracy but also their learning speed, which is crucial in settings such as hyperparameter optimization (including neural architecture search) or meta-learning. This mathematical framework unifies

different sub-fields of ML (*e.g.* transfer learning, meta-learning, ensemble learning), allows us to systematically classify methods, and provides us with formal tools to facilitate theoretical developments (*e.g.* the link to the No Free Lunch theorems) and future empirical research. In particular, it serves as the theoretical basis of a series of challenges that we organized.

Indeed, our principal methodological approach to tackle AutoML with Deep Learning has been to set up an extensive benchmark, in the context of a challenge series on Automated Deep Learning (AutoDL), coorganized with ChaLearn, Google, and 4Paradigm. These challenges provide a benchmark suite of baseline AutoML solutions with a repository of around 100 datasets (from all above domains), over half of which are released as public datasets to enable research on meta-learning. The challenge platform, the starting kit, the dataset formatting toolkit and all winning solutions are open-sourced. At the end of these challenges, we carried out extensive post-challenge analyses which revealed that: (1) winning solutions generalize to new unseen datasets, which validates progress towards universal AutoML solution; (2) Despite our effort to format all datasets uniformly to encourage generic solutions, the participants adopted specific workflows for each modality; (3) Any-time learning was addressed successfully, without sacrificing final performance; (4) Although some solutions improved over the provided baseline, it strongly influenced many; (5) Deep learning solutions dominated, but Neural Architecture Search was impractical within the time budget imposed. Most solutions relied on fixed-architecture pre-trained networks, with fine-tuning. Ablation studies revealed the importance of meta-learning, ensembling, and efficient data loading, while data-augmentation is not critical. All code and data (including post-challenge analyses data) are available at autodl.chalearn.org.

Besides the introduction of a novel general formulation of the AutoML problem, setting up and analyzing the AutoDL challenge, the contributions of this thesis include: (1) *Developing our own solutions to the problems we posed to the participants.* Our work *GramNAS* tackles the neural architecture search (NAS) problem by using a *formal grammar* to encode neural architectures. This provides a very robust and versatile solution to algorithm representation and opens the possibility to analyze learning of algorithms from the essential: after all, the ultimate representation of an algorithm is *its code* (together with a compiler). Two alternative approaches have been experimentally investigated: one based on Monte-Carlo Tree Search (MCTS) and one based on an evolutionary algorithm. As tree structures arise very naturally with formal grammars, Monte-Carlo Tree Search may be used rather naturally as search algorithm.

The MCTS GramNAS algorithm achieves state-of-the-art performance (94% accuracy) on CIFAR-10 dataset. We also cast on our GramNAS framework the AgEBO (Aging Evolution with Bayesian Optimisation) algorithm to illustrate the other approach. This last algorithm lends itself to parallelism. In a benchmark on 4 large well-known datasets, it beats state-of-the-art packages AutoGluon and AutoPytorch. The GramNAS framework provides insights to the understanding and representation of learning algorithms. A tool-kit was open-sourced to craft customized formal grammars for novel applications, allowing users to reuse common underlying search strategies. (2) *Laying the basis for a future challenge on meta-learning.* The AutoDL challenge series revealed the importance of meta-learning to succeed in solving AutoDL tasks. Yet the challenge setting did not evaluate meta-learning in the sense that meta-learning was not carried out on the challenge platform: code submitted by participants was only trained and tested independently on several tasks. With an intern, we are experimenting with various meta-learning challenge protocols. (3) *Making theoretical contributions.* During the course of this thesis, several collaborations were entered to tackle problems of meta-learning and transfer learning. We formulate meta-learning in a reinforcement learning setting and prove that under certain conditions, the average performance of the random search cannot be outperformed. We also make theoretical analysis on the super-generalization ability of the LEAP nets proposed by us and prove that when the perturbations of the system are additive, LEAP nets are capable of achieving super-generalization.

Abstract

(version française) L'apprentissage automatique automatisé (AutoML) vise à rendre l'application des méthodes d'apprentissage automatique (ML) aussi dépourvue d'intervention humaine que possible. Cet objectif ambitieux a fait l'objet de nombreuses recherches et techniques depuis les débuts du ML. L'objectif de cette thèse est de mettre un cadre formel autour de ce problème aux multiples facettes, de comparer les méthodes existantes et d'explorer de nouvelles directions.

Pour formuler le problème AutoML de manière rigoureuse, nous introduisons d'abord un cadre mathématique qui : (1) catégorise tous les algorithmes impliqués en trois niveaux (niveaux alpha, beta et gamma) ; (2) définit concrètement le concept de tâche (en particulier dans un cadre d'apprentissage supervisé) ; (3) définit formellement HPO et méta-apprentissage ; (4) introduit une métrique d'any-time learning qui permet d'évaluer les algorithmes d'apprentissage non seulement par leur précision, mais également par leur vitesse d'apprentissage. Ce cadre mathématique unifie différents sous-domaines du ML, nous permet de classer systématiquement les méthodes et nous fournit des outils formels pour faciliter les développements théoriques et de futures recherches empiriques. En particulier, il sert de base théorique à une série de challenges que nous avons organisés.

En effet, notre principale approche méthodologique pour aborder AutoML avec Deep Learning a été de mettre en place un benchmark étendu, dans le cadre d'une série de challenges sur l'Automated Deep Learning (AutoDL). Ces challenges fournissent une suite de référence de solutions AutoML de base avec un référentiel d'environ 100 datasets, dont plus de la moitié sont publiés sous forme de datasets publics pour permettre la recherche sur le méta-apprentissage. À la fin de ces challenges, nous avons effectué des analyses post-challenge approfondies qui ont révélé que : (1) les solutions gagnantes se généralisent à de nouveaux datasets invisibles, ce qui valide les progrès vers la solution universelle AutoML ; (2) Malgré nos efforts pour encourager des solutions génériques, les participants ont adopté des flux de travail spécifiques pour chaque modalité ; (3) L'any-time learning a été abordé avec succès, sans sacrifier la performance finale ; (4) Bien que certaines solutions se soient améliorées par rapport à la baseline fournie, elles en ont fortement influencé plusieurs ; (5) Les solutions d'apprentissage en profondeur dominaient, mais la

recherche d'architecture neuronale n'était pas pratique dans les délais impartis ; (6) Les études d'ablation ont révélé l'importance du méta-apprentissage, de l'assemblage et du chargement efficace des données, tandis que l'augmentation des données n'est pas critique. Tous les codes et données sont disponibles sur autodl.chalearn.org.

Outre l'introduction d'une nouvelle formulation générale du problème AutoML, la mise en place et l'analyse du challenge AutoDL, les contributions de cette thèse comprennent : (1) Développer nos propres solutions aux problèmes que nous avons posés aux participants. Notre travail GramNAS s'attaque au problème de la recherche d'architecture neuronale (NAS) en utilisant une grammaire formelle pour encoder les architectures neuronales. Deux stratégies de recherche alternatives ont été étudiées expérimentalement : une basée sur Monte-Carlo Tree Search (MCTS), qui atteint une précision de 94% sur le dataset CIFAR-10, et une autre basée sur un algorithme évolutif qui bat les packages de pointe AutoGluon et AutoPytorch sur 4 grands datasets bien connus ; (2) Former la base d'un futur challenge sur le méta-apprentissage ; (3) Au cours de cette thèse, plusieurs collaborations ont été engagées pour aborder des problèmes de meta-learning et transfer learning. Nous formulons le meta-learning dans un contexte d'apprentissage par renforcement et prouvons que dans certaines conditions, la performance moyenne de la recherche aléatoire ne peut pas être dépassée. Nous effectuons également une analyse théorique sur la capacité de super-généralisation des réseaux LEAP que nous proposons et prouvons que lorsque les perturbations du système sont additives, les réseaux LEAP sont capables de réaliser une super-généralisation.

Contents

List of Figures

List of Tables

Glossary

AS Algorithm Selection. 11, 12, 16

AutoDL Automated Deep Learning. 3, 7, 10, 25, 39, 43

AutoML Automated Machine Learning. 1, 9, 25, 108

DAG Directed Acyclic Graph. 26, 87–89, 92

FSL Few-Shot Learning. 125, 126

GramNAS Our proposed approach that uses formal grammars to encode search space of neural architecture search and combine with search strategies such as Monte-Carlo Tree Search. 17, 18, 32, 101, 103, 106, 110, 111, 168, 193

HPO Hyperparameter Optimization. 2, 9, 11, 12, 16, 22, 23, 25, 28, 29, 34, 38, 118

MCTS Monte-Carlo Tree Search. 16, 17, 79, 94–97, 101–105, 109–112, 194

NAS Neural Architecture Search. 11, 12, 15–19, 21–23, 25, 33, 34, 38, 77, 79, 82, 84, 89, 94, 95, 104, 108, 112, 126, 193

NN Neural Network. 7, 16, 17, 87

RL Reinforcement Learning. 11, 15–17, 34, 97, 115–120, 123, 126, 165

RNN Recurrent Neural Network. 8, 16

List of Symbols

The next list describes a list of symbols that will be later used within the body of this thesis.

Roman Symbols

α, α_i A predictor, also called an α-level algorithm, which takes an example as input and outputs a label.

β, β_i A learner also called an β-level algorithm, which takes a learning task T as input and outputs a predictor α.

γ, γ_i A meta-learner, also called a γ-level algorithm, which takes a meta-dataset $\mathfrak{D}_{tr}$ as input and outputs a learner β.

$\mathscr{A}(\mathscr{X}, \mathscr{Y})$ Set of all algorithms going from the input space $\mathscr{X}$ to the output space $\mathscr{Y}$.

$\mathscr{B}$ Set of learners. Can serve as the problem space in Rice's model on Algorithm Selection Problem.

$\mathscr{D}(X)$ Probability distribution on the set X.

$\mathscr{T}$ Task space, a set of learning tasks. Can serve as the problem space in Rice's model on Algorithm Selection Problem.

$\mathscr{X}$ Set of examples, often as the input space

$\mathscr{Y}$ Set of labels, often as the output space

$\mathfrak{D}, \mathfrak{D}_{tr}, \mathfrak{D}_{te}$ Meta-dataset of the form $\{T_i, \beta_i, R_i\}_{i=1}^n$. The index tr is for meta-training set and te is for meta-test set.

$\mathfrak{p}$ Performance model $\mathfrak{p} : \mathscr{T} \times \mathscr{B} \to \mathbb{R}^n$ in Rice's Algorithm Selection Problem.

D, D_{tr}, D_{te} Dataset of the form $\{x_i, y_i\}_{i=1}^n$. The index tr is for training dataset and te is for test dataset.

L Loss function $L : \mathscr{Y} \times \mathscr{Y} \to \mathbb{R}^+$

$R(\alpha; D)$ Risk function evaluated for an α-level algorithm α on the training/validation/test set D.

R, R_i Abbreviated risk function value. Also used for corresponding random variables.

T, T_i A learning task of the form $(D_{tr}, L; P)$.

x, x_i An example, often represented by a feature vector or a feature tensor

y, y_i A label (of an example), often represented by a vector of binary entries

1 - Background and Motivation

Machine learning (Bishop, 2006; Mitchell, 1997) leverages the power of data to automatically construct (or *learn*) algorithms. It is used as an powerful tool to solve real-world problems (especially in this age of *Big Data*) and furthermore, to attain some form of Artificial Intelligence (AI). As a sub-field of machine learning, Deep Learning (LeCun et al., 2015) demonstrates impressive performance when solving problems in computer vision (He et al., 2015; Krizhevsky et al., 2012), natural language processing (Devlin et al., 2018; Vaswani et al., 2017), speech recognition (Graves et al., 2013) and also in more classic settings with feature-based tabular data. However, just as 'classic' machine learning, deep learning suffers from the tedious trial-and-error process in model selection (*e.g.* constructing new neural network architectures) or tuning hyperparameters (*e.g.* learning rate, weight decay, batch size, filter size). To cope with this problem, Automated Machine Learning (AutoML) (Hutter et al., 2018) aims at automating such resource-consuming procedures and applying machine learning algorithms without any human intervention. This is equivalent to the following objective (*AutoML dream*):

> Find one single algorithm to solve all learning problems.

With this overarching goal, if AutoML materialized, it would bridge the supply-demand gap of data scientists and machine learning experts.

Historically, many efforts have been made to achieve this AutoML dream, both in academia and the private sector. In academia, AutoML challenges (Guyon et al., 2018) have been organized from 2015 through 2018 and were collocated with top machine learning conferences such as ICML and NeurIPS to motivate AutoML research in the machine learning community. The winning approaches from such prior challenges (e.g. Auto-sklearn (Feurer et al., 2015)) are now widely used both in research and in industry. More recently, interest in Neural Architecture Search (NAS) has exploded (Baker et al., 2017; Cai et al., 2018; Elsken et al., 2019; Liu et al., 2019a; Negrinho and Gordon, 2017b; Zoph and Le, 2016). On the industry side, many companies such as Microsoft (Fusi

et al., 2018) and Google are developing AutoML solutions. Google has launched their own AutoML platform (Cortes et al., 2017) powered by NAS (Pham et al., 2018; Real et al., 2017, 2020; Zoph and Le, 2016) and meta-learning (Finn et al., 2017, 2019). In this thesis, we consider how one can apply AutoML to automate *deep learning*, from both theoretical and empirical aspects. As AutoML is a relatively young field, diverse problems in both aspects remain to be solved.

Theoretical predictions rightfully indicate that, when there is no similarity between tasks and/or algorithms, no single learning algorithm can outperform all others (Wolpert, 2001, 1996; Wolpert and Macready, 1997). However, in real application scenarios, certain learning algorithms work consistently better than others on particular domains. For instance, convolutional neural networks (Lecun et al., 1998) demonstrate astonishing learning ability (in terms of training error *and* generalization error) for the image domain (Krizhevsky et al., 2012) while other learning algorithms work less well. This poses interesting theoretical questions on the problem of meta-generalization: whether one can select algorithms that will perform well on future (test) tasks based on their performance on on past (training) tasks. This is a problem we will touch upon in this thesis. Other questions are worthy of theoretical investigation, such as the trade-off between accuracy and speed of learning algorithms, especially when time-consuming algorithms such as model selection and hyperparameter optimization (HPO) are applied as part of the learning process. In the Neural Architecture Search community, some approaches can even cost up to 8000 GPU days (Zoph and Le, 2016). So we would like our algorithms to not only make accurate predictions, but also achieve good accuracy as fast as possible. Thus an ideal AutoML algorithm should be able to explore potentially better (hyper-)parameter choices, but also exploit already good ones to be more frugal. This explore-exploit trade-off plays a central role in AutoML and we will elaborate on it in this thesis.

From the practical and empirical aspects, we will analyze and test existing AutoML algorithms on a diverse set of datasets for comparison and benchmarking. We will review a repository of around 100 datasets that are formatted during this PhD and show extensive bench-

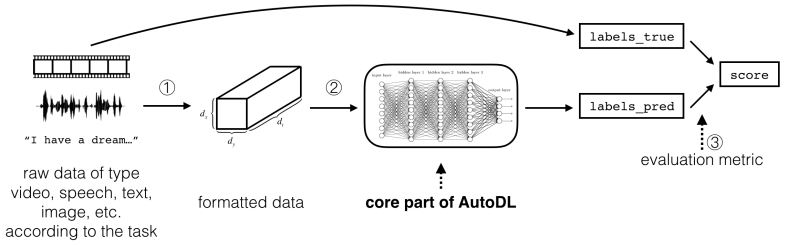


Figure 1.1 **Data flow in AutoDL challenges.** Different types (video, speech, text, etc) of data are first uniformly formatted in a tensor-based format then passed to the core part of AutoDL, which applies one single learning algorithm and tries to maximize the learning performance in terms of an evaluation metric

marking results. These datasets cover application domains such as computer vision, natural language processing, speech recognition and feature-based tabular data. Part of the results come from AutoDL challenges (Liu et al., 2021), a series of competitions we organized in the field of Automated Deep Learning (AutoDL). The problem to solve in these AutoDL challenges is in line with the aforementioned AutoML dream, as we can see from the data flow of AutoDL challenges shown in Fig. 1.1. From the figure, we see that all types (video, speech, text, etc) of data are first uniformly formatted in a tensor-based format. Then these examples are passed to the core part of AutoDL, which applies one single learning algorithm and tries to maximize the learning performance in terms of an evaluation metric.

As we realized how ambitious and challenging this goal of AutoDL can be, we divided the challenge into several smaller challenges, each concerning one domain/modality. There challenges are *AutoCV* (image), *AutoCV2* (image + video), *AutoNLP* (text), *AutoSpeech* (audio) and finally *AutoDL* (all combined). The participation and some basic facts of AutoDL challenges are shown in Table 1.1. These challenges clearly define what the AutoDL problem is, provide the community with an open-source benchmarking platform with a repository of 100 datasets, and help advance the state-of-the-art in this domain.

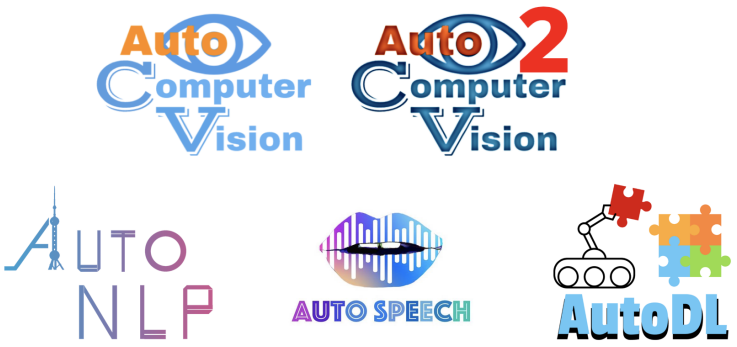


Figure 1.2 **Five competitions** in the AutoDL challenge series organized during the thesis: *AutoCV* (image), *AutoCV2* (image + video), *AutoNLP* (text), *AutoSpeech* (audio) and *AutoDL* (all combined).

Table 1.1 **Basic facts on AutoDL challenges**.

Challenge	Collocated with	Begin date 2019	End date 2019-20	#Teams	#Submissions
AutoCV	IJCNN	May 1	Jun 29	102	938
AutoCV2	ECML PKDD	Jul 2	Aug 20	34	336
AutoNLP	WAIC	Aug 2	Aug 31	66	420
AutoSpeech	ACML	Sep 16	Oct 16	33	234
AutoDL	NeurIPS	Dec 14	Mar 14	54	247

We couldn't make it happen without the help of many companies, institutes and conferences. Some of these collaborations are listed in Figure 1.3.

The general organization of this thesis is as follows.

— **Chapter 1** (this chapter) introduces the **background and motivation** of this work;

— **Chapter 2** reviews the **state-of-the-art** in the literature;

— **Chapter 3** clarifies the **scope of this work**, first introducing mathematical notations, then defining the empirical problems, and posing scientific questions addressed thereafter;

— **Chapter 4** presents the design, results and post-challenge analyses of the **AutoDL challenges**;

— **Chapter 5** develops our work on **Neural Architecture Search** (NAS). As a major distinguishing feature of AutoDL compared

Figure 1.3 **Collaborations during this thesis.** Google, 4Paradigm and Microsoft provide technical support and GPU credits. ChaLearn is the non-profit organization leading the organization of AutoDL challenges. Challenge results are shared in workshops collocated with conferences such as NeurIPS 2019, ECML PKDD 2019, ICLR 2020 and ICML 2020.

to AutoML in general, NAS aims to automate the choice of one important hyperparameter of deep learning models: the neural network architecture. This hyperparameter is different from more traditional hyperparameters since (1) it can be encoded by arbitrarily long strings and thus is discrete *and* of infinite choices; (2) it can encode much human knowledge (which could be one of the major reasons that contributes to the success of deep learning) that is hard to automate;

— **Chapter 6** introduces **meta-learning**, which is also an important component of more classic AutoML. Meta-learning, also known as *learning to learn*, tries to gain knowledge from past learning tasks and improve performance in future tasks. We lay the basis of the protocol for a future meta-learning challenge and propose baseline solutions. We also provide theoretical analysis on how promising meta-learning is useful when the hypotheses of the No Free Lunch theorems are not satisfied;

— Finally, we wrap up our work with **conclusions and lessons learned** in **Chapter 7**.

In Appendix, we attach some published papers in the course of this thesis.

2 - Related Work

As a relatively new domain, Automated Deep Learning (AutoDL) has its root from two domains of machine learning: Deep Learning and AutoML. In this chapter, we will review algorithms, which have recently emerged in these two domains. In Chapter 3 we will distinguish formally between three types of algorithms (also sometimes referred to as 'model'): (α) simple *predictor functions*, (β) *machine learning algorithm* (returning a predictor function), and (γ) *meta-learning algorithms* returning a chosen machine learning algorithm. In this chapter, we will use the world 'model' indistinguishably when there is no ambiguity on the type of algorithm, as is commonly done in the literature.

2.1 . Deep Learning

Deep learning is a sub-field of machine learning that uses computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These processing layers are often linear transformation, non-linearity activation or other modules such as pooling. The entire model is conventionally called a *neural network* (NN) (sometimes simply *neural net*). Deep learning has its root in the early 20th century (Hebb, 1949; Mcculloch and Pitts, 1943). Rosenblatt's 1957 paper on Perceptron (Rosenblatt, 1957) can be considered as a simple NN with no intermediate layers. Then more complicated and deeper networks such as multi-layer perceptron (MLP) and more complex network architectures were proposed in the 1970s and 80s (Rumelhart et al., 1986a,b) in Hinton's team. Bengio made several important contributions that helped promote 'deep' networks and forge the field (Bengio, 2009; Goodfellow et al., 2014). A brief survey and positional statement on deep learning in general can be found in (LeCun et al., 2015).

One of the most remarkable features of deep learning is its *flexibility* in designing network architectures, and the ease of training time by gradient descent. These architectures can encode much human

knowledge on specific domains, which allows (surprisingly) good performances in many applications. For instance, to attain invariance with respect to translation on images, *convolutional neural networks* were proposed in the late 1980's. They had great early success in character recognition (LeCun et al., 1989) and, more recently, in image classification (He et al., 2015; Krizhevsky et al., 2012). To model time series data and capture long-term dependencies between time frames (potentially non-Markovian), *recurrent neural networks* (RNN) have been proposed and deployed in speech recognition (Graves et al., 2014; Hochreiter and Schmidhuber, 1997; Mozer, 1992) and natural language processing (Devlin et al., 2018; Vaswani et al., 2017), though recent work seems to indicate that temporal convolutional networks (Waibel et al., 1989) might perform as well or better (Bai et al., 2018). To get an idea of how sophisticated neural network design gets, we borrow from (van Veen, 2017) which compiled following cheatsheet for neural networks developed in recent years, shown in Figure 2.1.

These are domains in which the network architecture plays a crucial role. With the maturity of hardware, the ubiquity of data and effective techniques for training, how to design good neural network architectures becomes one of the most important questions, for both industry and academia. In this thesis, one chapter (Chapter 5) will focus on approaching this problem in an automated manner.

2.2 . AutoML

The idea of AutoML could date back to the 1970s (Rice, 1976) with an accent on algorithm selection. Rice proposed to investigate a function (which we will call *performance model* in the future)

$$\mathfrak{p} : \mathscr{T} \times \mathscr{B} \to \mathbb{R}^n \tag{2.1}$$

where $\mathscr{T}$ is the problem space (e.g. set of learning tasks) and $\mathscr{B}$ is the algorithm space. The value space $\mathbb{R}^n$ is a performance measure/metric that judges the performance of applying an algorithm to a problem. It can be either a vector or a real number. To achieve AutoML, one can learn this performance model and use it to recommend good algo-

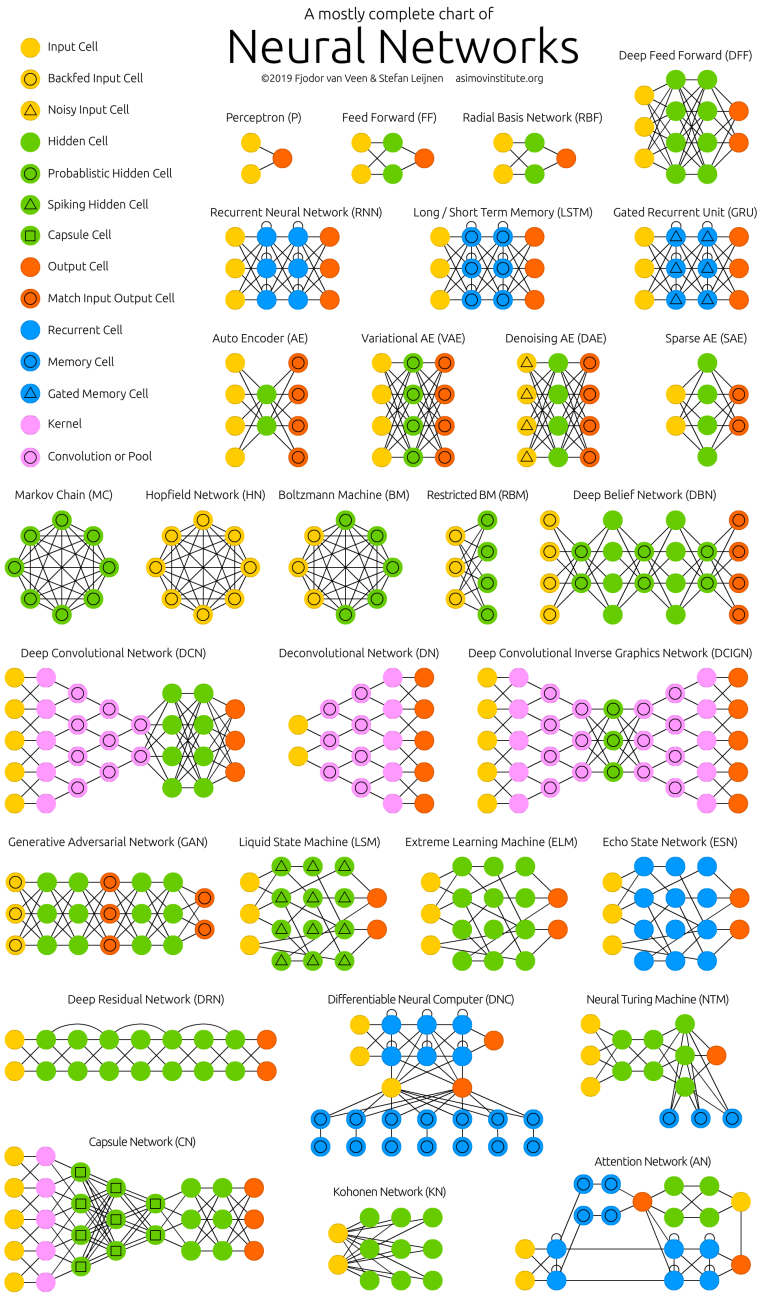


Figure 2.1 **Different types of neural networks** developed from 2004 to 2019. Figure taken from (van Veen, 2017). We see that the neural network family is getting more and more sophisticated and complex. The automation of designing neural network becomes very crucial.

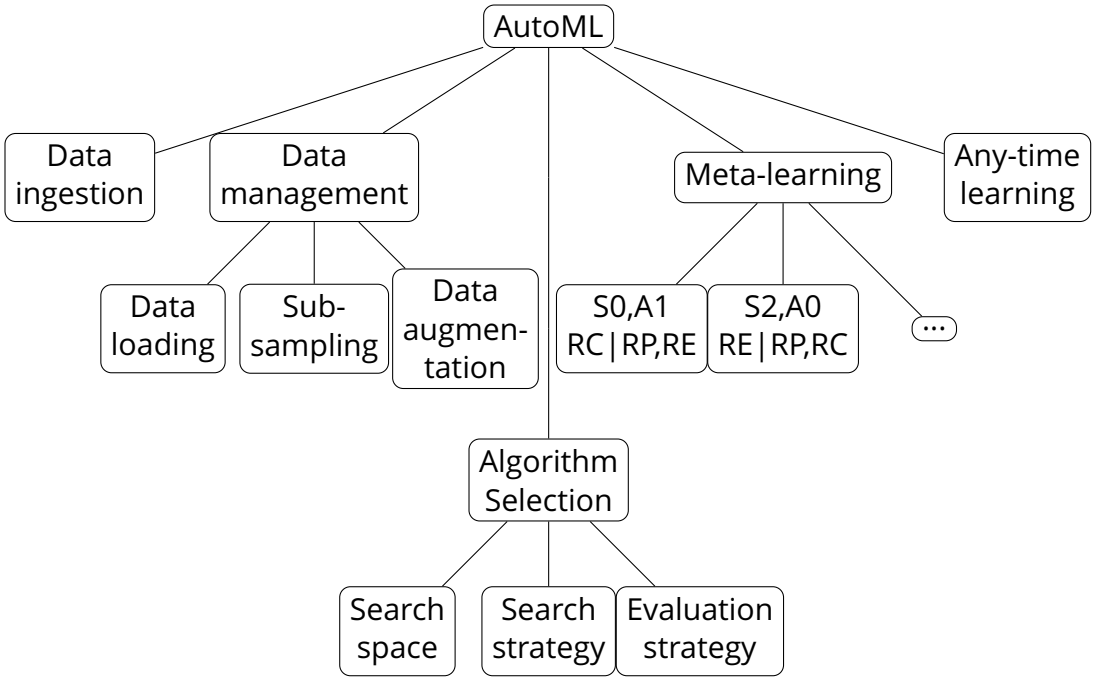


Figure 2.2 **Our taxonomy of AutoML.**

rithms for any new problems/tasks. In the 1980s, Schmidhuber (Schmidhuber, 1987) proposed a self-improving genetic programming system that can 'learn to learn', which can be considered as a form of meta-learning. In the 2000s, community efforts were focused on fighting against over-fitting using for example cross-validation (Bengio and Grandvalet, 2004; Kohavi, 1995) and many of these approaches can be seen as special cases of a 'multi-level' optimization framework (Guyon et al., 2011) In the 2010s, Bayesian optimization (Hutter et al., 2011a) has been introduced and the optimization aspect was emphasized. The term 'AutoML' was then coined by Hutter and a comprehensive survey can be found in (Hutter et al., 2018).

Depending on the specific sub-problem in AutoML, we come up with following taxonomy for AutoML (Figure 2.2).

— **Data ingestion** is about all steps we wish to automate before the learning step. This includes data collecting, pre-processing and anything that is typically done by a team of data engineers. We definitely want to automate data ingestion too but this subject is out of the scope of this thesis;

— **Data management** is about how one loads, samples and augments data (after the data are pre-processed) during the learning process. For example, one can adopt intelligent data loading strategy (e.g. to load a small batch of data first for learning) to achieve good any-time performance (Lim et al., 2019b; Liu et al., 2021). Data augmentation (Shorten and Khoshgoftaar, 2019) falls also into this category.

— **Algorithm selection** (AS) is about selecting an algorithm from a set of possible algorithms. A learning algorithm (often also called a *model* in machine learning community) is often encoded by its family (such as SVM or Random Forest) plus its hyperparameters (such as the kernel or number of estimators). Then Hyperparameter Optimization (HPO) and Neural Architecture Search (NAS) can be considered as special cases of algorithm selection. Algorithm selection will be an important focus of this thesis. Following (Hutter et al., 2018), we categorize the sub-problems of algorithm selection into: search space, search strategy and evaluation strategy;

— **Meta-learning** tries to make use of the knowledge from prior tasks to improve learning algorithms' performance in future tasks. Depending on which level of information is available, one may categorize meta-learning problems into 3 classes: 1) S0: zero-order meta-learning only has access to performances; 2) S1: first-order meta-learning additionally has access to (meta-)features of tasks and algorithms; and 3) S2:second-order meta-learning has access to full information to all tasks and algorithms. This categorization is very similar to Vanschoren's taxonomy (Vanschoren, 2018) on meta-learning which distinguishes methods that learn from *model evalution*, *task properties* or *prior models*. Based on the level of algorithms to evaluate, on can similarly have a categorization of A0, A1, etc. We will make more detailed discussion on this in Chapter 6 and *SAR tags* such as $S0, A1, RC|RP, RE$ and $S2, A0, RE|RP, RC$ will be introduced, within a reinforcement learning (RL) formulation;

— **Any-time learning**, or more generally *resources management*, is about delivering learning results under any resources (time, memory, etc) constraints. Typically, we don't want HPO or NAS approaches to try a huge number of models before delivering learning results. Instead, we wish to have an AutoML algorithm that is robust to time or memory constraints and can make predictions at any time with any computational resources.

In the following, we will focus on reviewing the literature for algorithm selection (HPO + NAS) and meta-learning, which are in line with the major contributions of this thesis.

2.2.1 . Hyperparameter Optimization & Algorithm Selection

Algorithm selection (AS) (Rice, 1976) aims at selecting an algorithm from a set of possible algorithms, based on data (or a performance metric). It has been an interesting and important problem since its birth. Hyperparameter Optimization (HPO) is a problem closely related to algorithm selection and can be considered as one of its subsets. Learning algorithms often come with some parameters, or *hyperparameters*, that need to be set in advance. How to choose these hyperparameters wisely and efficiently forms the central question of HPO. An HPO problem *is* an AS problem if one considers the algorithm specified by a hyperparameter as an independent algorithm. Otherwise, Thornton et al. (Thornton et al., 2012) still distinguishes them but consider a problem (the CASH problem) that merges algorithm selection and hyperparameter optimization. (Escalante et al., 2009) also considers a Full Model Selection (FMS) problem that deals with HPO (including selecting the algorithm *family*) and learning at the same time. We note that although we will focus on their application to machine learning, HPO and AS are also widely used in other domains such as solving satisfiability problem (SAT) (Hutter et al., 2002) and solving differential equations (Mezine et al., 2015).

In this work, we will mainly consider the case of supervised learning. Given a dataset of training data $D_{tr} = \{(x_i, y_i)\}_{i=1,\dots,n}$, a typical AS/HPO approach begins by splitting the training data further into two subsets: one forms the new training set, and another subset (or *validation set*) for estimating/validating the model's performance. A basic AS/HPO

method is to train every candidate model on the (new) training set, compute the performance measure using validation set then choose the model/hyperparameter with the highest performance. Such method is called *holdout validation*. To reduce the variance of the validation estimation, *cross validation* (Kohavi, 1995) method splits the training set into k (typically $k = 5$ or 10) equal pieces then validate the model k times using each piece as validation set and the other $k-1$ pieces as training set. k validation performances are then averaged to make a better estimation of the model performance. As the training of a model can be expensive in terms of computational resources, CV methods are very costly since they require k times resources. In addition, if the set of all candidate models is large , one cannot afford to validate all of them (even for holdout validation methods). Thus many methods are then proposed to remedy this problem.

Before explaining other HPO methods, we introduce some mathematical notations for convenience. We recall Rice's performance model in Equation (2.1) but only consider the real-valued case (i.e. for $n = 1$):

$$\mathfrak{p} : \mathscr{T} \times \mathscr{B} \to \mathbb{R} \tag{2.2}$$

where again $\mathscr{T}$ is the problem space, $\mathscr{B}$ is the algorithm space and the performance (such as the accuracy) is a real number. Most HPO methods restrict themselves to considering one single problem/task at a time, i.e. $\mathscr{T} = \{T_0\}$ is a singleton (note that this restriction is actually what distinguishes HPO from meta-learning). As for the algorithm space, $\mathscr{B}$ often involves both continuous *and* discrete hyperparameters. Typically, the set of all algorithm families (such as SVM or Random Forest) is discrete. This set is also *conditional* in the sense that other hyperparameters (whether the value or the existence) might rely on the choice of it. We now introduce some HPO approaches with these notations.

To avoid trying out every single candidate model (and then just choosing the best in the end) such as *grid search*, Bergstra and Bengio (2012) shows that randomly choosing hyperparameters (for only a few shots), or *random search*, already forms a surprisingly strong baseline for HPO. Otherwise, many approaches exploit a certain *structure* on

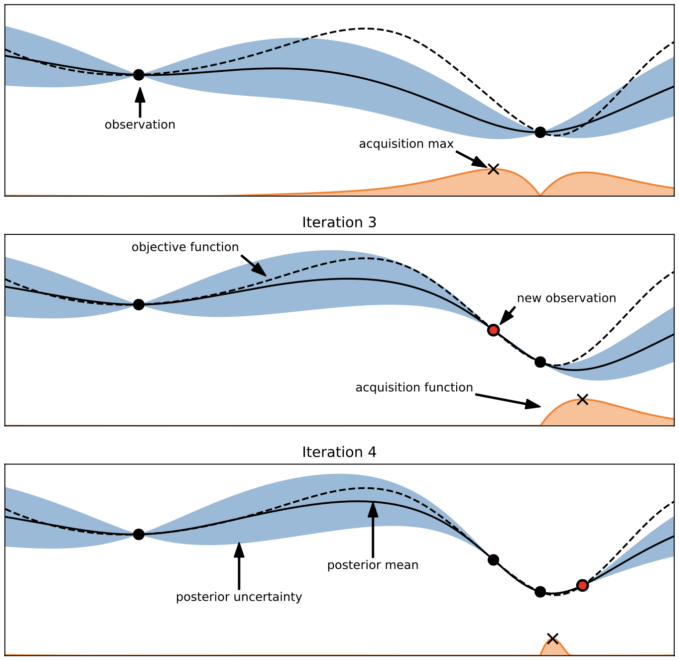


Figure 2.3 **Bayesian optimization on real-valued functions.** Figure from (Hutter et al., 2018).

the set of all candidate models, e.g. the correlation of performances between different models. This allows reducing the computational cost while still gaining an appropriate amount of information for a better performance estimation.

Bayesian Optimization (Snoek et al., 2012) methods tries to infer this structure by setting a *prior* on the performance model $\mathfrak{p} : \{T_0\} \times \mathscr{B} \to \mathbb{R}$ (or simply some $f : \mathscr{B} \to \mathbb{R}$ as the task is clear) and progressively gaining knowledge on this function. The performances of two algorithms/hyperparameters will be correlated when they are 'close' in certain sense. Some first HPO methods use Gaussian Processes (Snoek et al., 2012) in which all function values (on any finite subset of the hyperparameter space) follow a multivariate Gaussian distribution. However, estimating the performance model using Gaussian Process suffers from several limitations: 1) one often sets $\mathscr{B} = [0, 1]^{n_{HP}}$ (with n_{HP} the number of hyperparameters) and this cannot deal with discrete hyperparameters well; 2) one cannot deal with conditional hyperparameters well; 3)

it is of cubic complexity $O(n_{HP}^3)$ and thus not scalable to the case with many hyperparameters.

For better scalability in higher dimension, Hutter et al. (Hutter et al., 2011a) use random forest instead of GP to model the performance function (and this becomes later a sub-routine of the competition winner solution Auto-sklearn (Feurer et al., 2015)). Another BO-based approach, TPE (Tree-structured Parzen Estimator) (Bergstra et al., 2011) also uses tree-based models to infer the inverse prior (i.e. $p(x|y)$ instead of $p(y|x)$, where x is the hyperparameter and y is the performance). Budget-aware BO methods such as (Klein et al., 2017) have been proposed to take the computational cost into consideration and be time efficient. Then it's worth noting that BO methods have also been used for designing neural network architectures (Snoek et al., 2015).

Besides BO methods, *evolutionary algorithms* (EA) form another family of solutions for HPO. In the EA philosophy, hyperparameters/algorithms are considered as *individuals* and a *set* of individuals (the *population*) is considered at the same time, instead of one single individual as in the case of BO. Thus one can say that EA optimize with respect to a *distribution* instead of a point. EA approaches model the aforementioned structure by the continuity of performance among 'close' individuals, related by small changes (i.e. *mutations*). Escalante's Particle Swarm method falls into this category. Later we will see more EA algorithms in the section of NAS.

Reinforcement learning (RL) provides another path to solve HPO. Approaches using RL begin by formulating the HPO problem (which is essentially an optimization problem) as an RL problem. Typically, the actions are choosing and evaluating a hyperparameter, the reward is the obtained validation accuracy and the states are current hyperparameter configurations (or simply stateless). After this re-formulation, many RL algorithms can be automatically applied to HPO. As an example, Li et al.'s Hyperband (Li et al., 2016) formulates the HPO problem as a stateless RL problem and applies multi-armed bandit based algorithm. Then later Falkner et al. (Falkner et al.) combines this approach with BO methods.

Besides all above approaches, differentiable methods using a bi-level formulation such as (Bennett et al., 2006; Franceschi et al., 2018) also exist.

2.2.2 . Neural Architecture Search

Neural architecture search (NAS) can be regarded as a special form of algorithm selection or of HPO. Indeed, it concerns particularly the family of neural networks, which is only a subset of all machine learning algorithms. However, with the recent success of deep learning, NAS has gained explosive attention from the research community in the late 2010s. Due to the specialty of neural network architectures, a rather independent literature has been formed and it is worth it to have a separate introduction for NAS. We note that early works such as NEAT (Miikkulainen, 2002) also search for NN architectures. NEAT optimizes with respect to the weight parameters AND the architecture at the same time, while recent works focus on the architecture part in most cases. Although most recent surveys on NAS (Elsken et al., 2019; Ren et al., 2020; Wistuba et al., 2019) focus on works since 2016, many earlier works can be regarded as ancestors of this domain.

From the deep learning literature review above, we can see that the main focus of many works (He et al., 2015; Hochreiter and Schmidhuber, 1997; Krizhevsky et al., 2012; Szegedy et al., 2015) is proposing a new neural network architecture. This process of manually designing architectures is often a long trial-and-error process, which could be even more tedious than the case of classic machine learning, since the training of deep learning models require much more time in general. NAS methods were proposed to automate this process, which obviously is a special case of AS/HPO.

One of the first works in NAS literature is Zoph and Le's 2016 paper (Zoph and Le, 2016), which proposes an RL framework for automatically designing NN architectures. In this framework, architectures are designed in a sequential manner. Each action chooses a configuration (e.g. convolutional layer or pooling) for next layer and states are the current configuration of all previous layers. The reward feedback is the accuracy when the architecture is complete (or zero otherwise). An RNN controller then is applied to learn a policy of designing architectures,

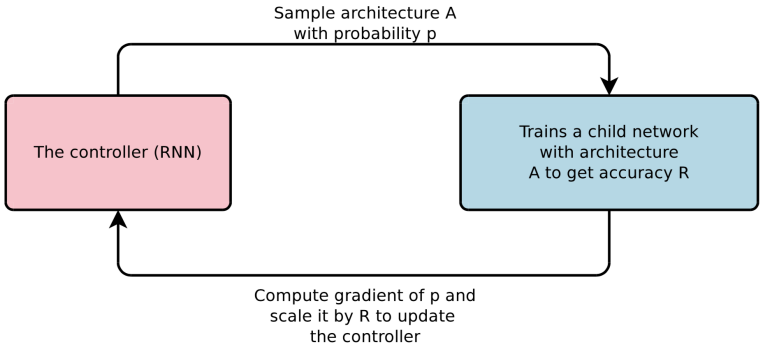


Figure 2.4 **Reinforcement learning approach used in one of the first NAS works.** Figure from (Zoph et al., 2018). States consist of architectures up to current layer. Actions are choices for next layer. The reward is the accuracy obtained if the architecture is complete and 0 otherwise.

using policy gradient (Sutton and Barto, 2018). This approach managed to design a NN architecture that achieves 96.5% accuracy on the CIFAR-10 (Krizhevsky, 2009) dataset, which was the state of the art back then. However this approach consumes huge amount of computational resource (28 days on 800 GPUs). In the same year, Baker et al. (Baker et al., 2017) proposed a similar approach also with an RL formulation, which instead uses Q-learning with ε-greedy (Sutton and Barto, 2018) method. This approach has a training phase of 8 to 10 days with 10 GPUs. Evolution strategies have been used (Real et al., 2017) for NAS, which allows parallelization but still needs much resources. By reviewing the literature extensively, we came up with Table 2.1 which lists a few NAS approaches with their corresponding computational cost and we can see that most NAS methods require a considerable amount of GPU cost.

To remedy the problem of huge computational cost, many NAS methods are proposed later. NASNet (Zoph et al., 2018) proposes a particular search space with repeated layers and modules, which dramatically reduces the search space and saves computational cost. Many later works (Liu et al., 2018a, 2019a; Pham et al., 2018; Wang et al., 2018) followed this idea and adopted the same search space. An important step for reducing the search time in the NAS literature is the use of *weight sharing* (or *warm start* to not be confused by the concept

Table 2.1 **Computational resources required in state-of-the art NAS works.** We list several AutoDL methods with their consumed computational resources for running their experiments. The equivalent amount of GPU days on an Nvidia Tesla V100 card is computed in the last bold column. Note that these amounts of computing resources correspond to only **a single run** of the experiment. The work behind is usually multiplied by a factor of at least 3.

Methods	GPU Model (Nvidia)	# GPU	Duration /hours	exaFLOP (10^{18} FLOPs)	GPU·hour (V100)	**GPU·day (V100)**
NAS (2016)	Tesla K40	800	672	9754.21	181858.1	**7577.4**
MetaQNN (2017)	Tesla K40	10	240	43.55	811.9	**33.8**
ENAS (2018)	GTX 1080Ti	1	16	0.65	12.2	**0.5**
PNAS (2018a)	Tesla P100	100	96	174.18	3247.5	**135.3**
LSE (2017)	Tesla K40	1000	300	5443.20	101483.3	**4228.5**
DARTS (2018b)	GTX 1080Ti	4	24	3.92	73.1	**3.0**
NASNet (2018)	Tesla K40	500	96	870.91	16237.3	**676.6**
EAS (2017)	GTX 1080	5	48	7.67	142.9	**6.0**
AlphaX (2018)	GTX 1080	17	120	65.16	1214.9	**50.6**

of weight sharing in the case of convolutional neural networks), which reuses the weights in previously trained architectures instead of training each architecture from scratch each time. ENAS (Pham et al., 2018) proposes this idea and costs only 16 hours on one single GPU. More later approaches followed this idea of weight sharing (Chu et al., 2020; Zhang et al., 2020). Another path that is worth mentioning is the use of relaxation of hard choices of configurations and uses gradient-based methods for NAS. DARTS (Liu et al., 2019a) uses a probabilistic approach to relax the (hard) choices of configurations (such as convolutional or max-pooling) to a softmax combination on these choices. Then one can use gradient-descent to train the softmax coefficients using validation data. This approach can find reasonably good architectures in a one-day time window with one single GPU.

2.2.3 . Meta-learning

In contrast to HPO where only the data of the current task is exploited, meta-learning tries to make use of the knowledge from *prior* tasks to improve learning algorithms' performance in future tasks. This improvement of performance can be in terms of either the computa-

tional aspect (e.g. in an *any-time learning* setting) or statistical aspect (e.g. in a *few-shot learning* setting). As stated previously, meta-learning can date back to the 1970s (Rice, 1976) and 1980s (Schmidhuber, 1987). Systematic survey work exists as early as 2008 (Brazdil et al., 2008).

For meta-learning, there is also a form of the previously mentioned *structure*. This time, it exists in the relationship among different tasks and could be interpreted as the *similarity* between tasks. If two tasks are by no means 'similar', it would be hard to apply useful knowledge from one task to another.

Very often, characterizing the similarity between tasks reduces to finding appropriate task *representation*. Typically, this task representation could be a set of (hand-crafted) meta-features (e.g. number of examples, shape of images) computed for each task. Then algorithms from supervised learning or recommender systems (Bobadilla et al., 2013; Mısır and Sebag, 2017) can be applied to recommend promising learning algorithms (model or hyperparameters). This is the approach used in, for instance, the meta-learning step of Auto-Sklearn (Feurer et al., 2015) and (Muñoz et al., 2018). Note that such approaches could use not only a task representation but also an algorithm representation for learning algorithms. But since HPO naturally adopts such an algorithm representation (typically an algorithm family plus the values of hyperparameters), we can directly borrow ideas from HPO literature. Besides above approaches using explicit task representation and/or algorithm representation, there also exist methods that are only based on performance data (i.e. past performances of applying algorithms to tasks). (Sun-Hosoya, 2019) and (Mısır and Sebag, 2017) borrow ideas from recommendation systems and apply them to the performance data. (Van Rijn and Hutter, 2018) analyzes performances data and evaluate the importance of hyperparameter, which gives an interesting example of applying meta-learning to HPO.

Above works concern mostly tabular datasets (e.g. from OpenML (Vanschoren et al., 2014)) and the computational aspect of meta-learning for accelerating HPO. There also exist approaches discussing the statistical aspect of meta-learning under the form of few-shot learning. In few-shot learning, one wishes to solve classification tasks with only a few

(e.g. one) examples per class, by borrowing knowledge from prior tasks. An important amount of works in this direction deals with image classification tasks. Instead of learning a representation per *task*, MAML (Finn et al., 2017) tries to learn a convenient common representation for all *examples* across all tasks. The learning follows a gradient-based bi-level formulation scheme and the common representation is updated based on a loss function on the validation set. Similar ideas with bi-level formulation can be found in (Franceschi et al., 2018; Liu et al., 2019a). More review and discussion on few-shot learning will be presented in Chapter 6 of meta-learning.

In Chapter 6, we will also give a reinforcement learning formulation for meta-learning, which allows us to classify meta-learning approaches conveniently and systematically.

2.2.4 . Challenges and Benchmarks

To fairly evaluate AutoML/AutoDL methods and help enhance community efforts towards reproducibility (Pineau et al., 2020), several challenges and benchmarks were organized. A first example is OpenML (Van Rijn et al., 2013; Vanschoren et al., 2014), which we already mentioned above. OpenML is a machine learning benchmarking platform that consists of (as of 2020) more than 20,000 tabular datasets and more than 15,000 algorithms (called flows). Performances of algorithms executed on different datasets/tasks are recorded in the database. Open-source API's are provided to open the access of these results, in the purpose of enabling meta-learning. Figure 2.5 from (Sun-Hosoya, 2019) shows a hierarchical clustering of dataset-algorithm performances for a subset of OpenML. We see very clear block structure in the figure. Thus certain 'structure' does exist in this meta-dataset, opening the possibility of meta-learning.

Apart from OpenML, Guyon et al. (Guyon et al., 2015, 2018) organized a series of AutoML challenges from 2015 to 2018, where tabular datasets defining 30 supervised learning (classification and regression) tasks are provided. Figure 2.6 shows some statistics of the datasets used in these AutoML challenges. These datasets cover diverse domains (finance, medical, digits, etc) and vary a lot in terms of number of training/validation/test examples (Ptr, Pva and Pte respectively). The task

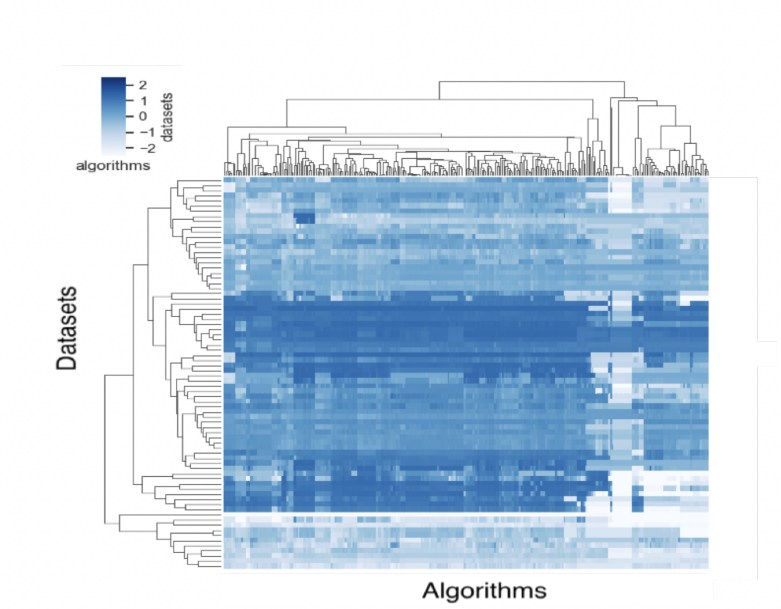


Figure 2.5 **Hierarchical clustering of dataset-algorithm performances for the OpenML meta-dataset (Sun-Hosoya, 2019).** A subset of 292 datasets of OpenML (Vanschoren et al., 2014) is considered. Performances of 76 learning algorithms are recorded, making the shape of the performance matrix 76×292.

(multilabel, regression, etc) was also different from dataset to dataset. Some aggregated results of AutoML challenges are shown in Figure 2.7. We see that the modeling difficulty (i.e. the variance of performances among different approaches) of datasets in AutoML challenges varies a lot, showing how challenging the AutoML problem is and how robust an AutoML algorithm should be.

The top-1 AutoML challenges winner's solution Auto-sklearn (Feurer et al., 2015) (Figure 2.8) combines meta-learning, Bayesian optimization and ensemble methods (Caruana et al., 2004) and is widely used in both academia and industry.

Benchmarks focusing on NAS were proposed recently too. NAS-Bench-101 (Ying et al., 2019) constructed a NAS benchmark on the CIFAR-10 (Krizhevsky, 2009) dataset, with a particularly crafted search space inspired by NASNet (Zoph et al., 2018) search space. This benchmark provides a quick way to benchmark NAS algorithms on an exhaustively pre-computed performance dataset, which is important since it

Round	Num	Name	Task	Metric	Time	Cnum	Cbal	Sparse	Missng	Catvar	Irrvar	Pte	Pva	Ptr	N	Ptr/N
0	1	ADULT	multilabel	F1	300	3	1	0.16	0.011	1	0.5	9768	4884	34190	24	1,424.58
0	2	CADATA	regression	R2	200	0	NaN	0	0	0	0.5	10640	5000	5000	16	312.5
0	3	DIGITS	multiclass	BAC	300	10	1	0.42	0	0	0.5	35000	20000	15000	1568	9.57
0	4	DOROTHEA	binary	AUC	100	2	0.46	0.99	0	0	0.5	800	350	800	100000	0.01
0	5	NEWSGROUPS	multiclass	PAC	300	20	1	1	0	0	0	3755	1877	13142	61188	0.21
1	1	CHRISTINE	binary	BAC	1200	2	1	0.071	0	0	0.5	2084	834	5418	1636	3.31
1	2	JASMINE	binary	BAC	1200	2	1	0.78	0	0	0.5	1756	526	2984	144	20.72
1	3	MADELINE	binary	BAC	1200	2	1	1.2 E-06	0	0	0.92	3240	1080	3140	259	12.12
1	4	PHILIPPINE	binary	BAC	1200	2	1	0.0012	0	0	0.5	4664	1166	5832	308	18.94
1	5	SYLVINE	binary	BAC	1200	2	1	0.01	0	0	0.5	10244	5124	5124	20	256.2
2	1	ALBERT	binary	F1	1200	2	1	0.049	0.14	1	0.5	51048	25526	425240	78	5,451.79
2	2	DILBERT	multiclass	PAC	1200	5	1	0	0	0	0.16	9720	4860	10000	2000	5
2	3	FABERT	multiclass	PAC	1200	7	0.96	0.99	0	0	0.5	2354	1177	8237	800	10.3
2	4	ROBERT	multiclass	BAC	1200	10	1	0.01	0	0	0	5000	2000	10000	7200	1.39
2	5	VOLKERT	multiclass	PAC	1200	10	0.89	0.34	0	0	0	7000	3500	58310	180	323.94
3	1	ALEXIS	multilabel	AUC	1200	18	0.92	0.98	0	0	0	15569	7784	54491	5000	10.9
3	2	DIONIS	multiclass	BAC	1200	355	1	0.11	0	0	0	12000	6000	416188	60	6,936.47
3	3	GRIGORIS	multilabel	AUC	1200	91	0.87	1	0	0	0	9920	6486	45400	301561	0.15
3	4	JANNIS	multiclass	BAC	1200	4	0.8	7.3 E-05	0	0	0.5	9851	4926	83733	54	1,550.61
3	5	WALLIS	multiclass	AUC	1200	11	0.91	1	0	0	0	8196	4098	10000	193731	0.05
4	1	EVITA	binary	AUC	1200	2	0.21	0.91	0	0	0.46	14000	8000	20000	3000	6.67
4	2	FLORA	regression	ABS	1200	0	NaN	0.99	0	0	0.25	2000	2000	15000	200000	0.08
4	3	HELENA	multiclass	BAC	1200	100	0.9	6 E-05	0	0	0	18628	9314	65196	27	2,414.67
4	4	TANIA	multilabel	PAC	1200	95	0.79	1	0	0	0	44635	22514	157599	47236	3.34
4	5	YOLANDA	regression	R2	1200	0	NaN	1 E-07	0	0	0.1	30000	30000	400000	100	4000
5	1	ARTURO	multiclass	F1	1200	20	1	0.82	0	0	0.5	2733	1366	9565	400	23.91
5	2	CARLO	binary	PAC	1200	2	0.097	0.0027	0	0	0.5	10000	10000	50000	1070	46.73
5	3	MARCO	multilabel	AUC	1200	180	0.76	0.99	0	0	0	20482	20482	163860	15299	10.71
5	4	PABLO	regression	ABS	1200	0	NaN	0.11	0	0	0.5	23565	23565	188524	120	1,571.03
5	5	WALDO	multiclass	BAC	1200	4	1	0.029	0	1	0.5	2430	2430	19439	270	72

Figure 2.6 Statistics of datasets used in AutoML challenges (Guyon et al., 2015).

could be very hard to evaluate NAS algorithms from scratch (Yang et al., 2020). Similar benchmarks such as NAS-Bench-1Shot1 (Zela et al., 2020) and NAS-Bench-201 (Dong and Yang, 2020) extend this benchmark.

Besides works in the machine learning community, competitions in algorithm selection (Lindauer et al., 2017, 2019) have also been organized recently and are highly relevant.

2.2.5 . Prior work highlights and open problems

In view of the literature review, we identified a number of research directions worthy of our attention.

— **Gaining a unified view for HPO, NAS and meta-learning.** Works in the literature often treat these three domains separately. However, what is the relationship between them and can one gain a unified view for them? We will discuss this in Chapter 3;

— **Fairly comparing AutoDL methods.** Comparing AutoDL methods such as HPO and NAS is not easy. We will introduce in Chapter 4 a series of competitions, the AutoDL challenges, which aim to fairly comparing AutoDL methods. In Chapter 6, we will also introduce the MetaDL challenge that has a focus on few-shot learning;

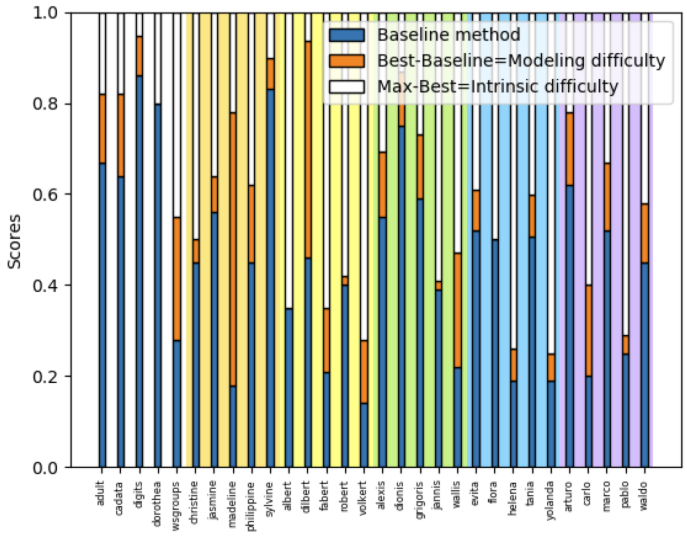


Figure 2.7 **Results of participants' solutions from AutoML challenges (Guyon et al., 2018).** For each dataset, the blue part shows the baseline performance. The orange part shows the performance difference between baseline and best participant solution. The white part is the difference between the maximum performance (which 1.0 in this case) and the best participant. The modeling difficulty of datasets in AutoML challenges varies a lot, showing how challenging the AutoML problem is and how robust an AutoML algorithm should be.

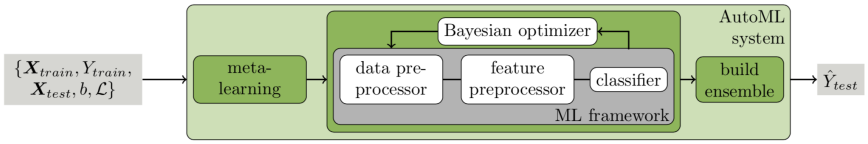


Figure 2.8 **Pipeline of Auto-sklearn**(figure from (Feurer et al., 2015)), the winner of AutoML challenges (Guyon et al., 2015). Auto-sklearn combines meta-learning, Bayesian optimization and ensemble methods.

— **Finding efficient algorithm representations.** Using a good algorithm representation is crucial to many HPO and NAS methods. We will propose an algorithm representation with formal grammar in Chapter 5;

3 - Scope of Work

In this chapter, we formulate the problems treated in this thesis and describe the scope of work. First, we give a mathematical formulation of the problem of meta-learning (Section 3.1), then we describe in that setting the meta-learning problems addressed in this thesis (Section 3.2), and finally we outline the contributions made in this thesis (Section 3.3).

3.1 . Mathematical Problem Formulation

The central problem of this thesis is Automated Deep Learning (AutoDL). In this section, we frame the problem in the larger context of Automated Machine Learning (AutoML), and bring it back to that of *meta-learning*. We first position this problem in the more general scope of "algorithm selection" (Rice's setting), which concerns solving problems outside of machine learning, then specialize it to meta-learning. We introduce a hierarchy of sub-problems. This will allow us, in Section 3.2, to position the main themes of this thesis, including Hyperparameter Optimization (HPO) and Neural Architecture Search (NAS), determining a challenge winner, and active meta-learning.

3.1.1 . Rice's Algorithm Selection Problem

In his 1975 paper, Rice formulated the *Algorithm Selection Problem*, of which meta-learning is a special case, and introduced the following abstract model (Rice, 1976). Consider a problem space $\mathscr{T}$, an algorithm space $\mathscr{B}$ and a function $\mathfrak{p}$ called the **performance model**:

$$\mathfrak{p} : \mathscr{T} \times \mathscr{B} \to \mathbb{R}^n, \tag{3.1}$$

where $\mathfrak{p}(T, \beta)$ (for $T \in \mathscr{T}$ and $\beta \in \mathscr{B}$) is the performance measure obtained when applying the algorithm β on the problem T. Then the **Algorithm Selection Problem** is: to determine a **selection mapping**

$$S : \mathscr{T} \to \mathscr{B} \tag{3.2}$$

that verifies certain optimality criterion. For instance, one example of such optimality criterion given by Rice is called 'Best Selection', which chooses the selection mapping S to be certain oracle function that verifies [1]

$$\|\mathfrak{p}(T, S(T))\| \leq \|\mathfrak{p}(T, \beta)\|, \quad \text{for any } \beta \in \mathscr{B} \text{ (and any } T \in \mathscr{T}), \tag{3.3}$$

where $\|\cdot\|$ is a norm on $\mathbb{R}^n$.

As we shall see in the following, many problems treated in this thesis such as neural architecture search and meta-learning are some form of the Algorithm Selection Problem, which can thus serve as a central piece that relates all future chapters. In this thesis, we will only consider the case $n = 1$ and directly treat $\mathfrak{p}(T, \beta)$ as an algorithm performance in $\mathbb{R}$ (instead of a multi-dimensional 'performance measure' in $\mathbb{R}^n$).

When the performance model $\mathfrak{p}$ is known and can by easily computed, the Algorithm Selection Problem can be simply solved by the following selection mapping

$$\boxed{S(T) := \underset{\beta \in \mathscr{B}}{\arg\min}\, \mathfrak{p}(T, \beta).} \tag{3.4}$$

However, this solution is almost never realizable in practice, since every component of $\mathfrak{p}$, T and β can be very complex, see next section. We also note here that the right-hand side of (3.4) is exactly what the 'Best Selection' criterion in (3.3) demands.

We will now specialize Rice's framework in two respects to relate it to meta-learning:

1. define machine learning tasks $\mathscr{T}$ and algorithms $\mathscr{B}$,
2. discuss how performances are obtained in machine learning, using empirical data.

3.1.2 . Problem Space for Machine Learning

First we consider specializing the space of tasks $\mathscr{T}$ (called "problem space" by Rice).

1. In this chapter we suppose that the performance model $\mathfrak{p}$ corresponds to 'negative performance', i.e. the smaller the better.

In the field of machine learning, the tasks in the problem space $\mathscr{T}$ take particular forms. As we will focus on *supervised* learning scenarios in this work, we will consider learning tasks $T \in \mathscr{T}$ taking the form

$$T = (D_{tr}, L; P),$$

composed of a training dataset of n examples $D_{tr} = \{(x_i, y_i)\}_{i=1}^{n} \subseteq (\mathscr{X} \times \mathscr{Y})^n$, a loss function L and an *unknown* underlying distribution P on $\mathscr{X} \times \mathscr{Y}$ (the semi-colon separates the known arguments and the unknown/hidden arguments). We further assume that the examples are *i.i.d.* realizations $(x_i, y_i) \overset{iid}{\sim} P$ and $L(y', y)$ is a function measuring the discrepancy between a prediction y' and a target value y. Given the task T, we look for an algorithm α that can predict a label y as correct as possible given an example x. That is, α has the following signature

$$\alpha : x \mapsto y. \tag{3.5}$$

We will call such an algorithm α a **predictor** (also commonly called a *model* and sometimes called an α**-level algorithm** by us). We will also say that such an algorithm is **in** α**-level**. The **expected risk** (or *generalization error*) measuring the expected performance of a predictor α on the task T is defined to be

$$R(\alpha) := \int L(\alpha(x), y)\, dP(x, y)\ . \tag{3.6}$$

The goal of the learning task T is to *find a predictor* α^* *that minimizes the expected risk*. That is, we wish to solve the following supervised learning problem

$$\alpha^* = \underset{\alpha \in \mathscr{A}(\mathscr{X}, \mathscr{Y})}{\arg\min}\ R(\alpha) \tag{3.7}$$

where $\mathscr{A}(\mathscr{X}, \mathscr{Y}) \subseteq \mathscr{Y}^{\mathscr{X}}$, which we will call a **predictor set**, is the set of *all* possible algorithms that go from $\mathscr{X}$ to $\mathscr{Y}$.

Here one should note the difference between a predictor set and a *hypothesis set*. A hypothesis set is a choice of the learning algorithm and is often a small subclass of the predictor set, while a predictor set

is a theoretical set that only depends on the input space $\mathscr{X}$ and the output space $\mathscr{Y}$.

3.1.3 . Algorithm Space for Machine Learning

Second we specialize the space of learning algorithms considered $\mathscr{B}$ (called "algorithm space" by Rice).

Algorithm selection, in application to machine-learning corresponds to selecting an algorithm β (which in turn returns a predictor α, Eq. 3.5). Formally, the goal of a learning task

$$T = (D_{tr}, L; P)$$

is to find a predictor α that minimizes the expected risk. An algorithm β (which we will call a **learner**, or a β**-level algorithm**) solving this task should thus have the following signature:

$$\beta : T \mapsto \alpha \ . \tag{3.8}$$

Following Eq. 3.6, the performance model $\mathfrak{p}$ for machine learning can be defined as

$$\mathfrak{p}(T, \beta) = \int L(\beta(T)(x), y) \, dP(x, y). \tag{3.9}$$

We recall that the components L and P come with the task T and are specific to this task. Also, as P is unknown to β, the learner β can only make use of D_{tr} and L.

3.1.4 . Two-level learner

In this section, we introduce a particular case of learner, called two-level learner, often identified in machine learning with the process of performing both hyperparameter selection (and/or model selection) and parameter optimization.

We recall that the goal of the Algorithm Selection Problem is to find a selection mapping

$$S : \mathscr{T} \to \mathscr{B} \tag{3.10}$$

that verifies for example the 'Best Selection' optimality criterion.

If one thinks of such S as an HPO algorithm (e.g. grid search or random search with cross-validation (Bergstra and Bengio, 2012) for

SVM (Boser et al., 1992)) one can further define a two-level learner β_S by

$$\beta_S : T \mapsto \alpha = S(T)(T), \tag{3.11}$$

which chains HPO and parameter optimization. For instance, one can think of T as the task corresponding to the MNIST dataset (Lecun et al., 1998) and S as the HPO algorithm that finds an optimal set of hyperparameters of the SVM algorithm from T, thus the SVM with these hyperpameters correspond to the *learner* $S(T)$ (higher-level optimization). Then what β_S does is nothing but applying the found SVM $S(T)$ on the MNIST task T to adjust the SVM parameters (lower level optimization). The obtained predictor $S(T)(T)$ is then used for predictions. The whole process going from a task T to a predictor $S(T)(T)$ defines a learner, which only depends on S.

We have thus brought back any selection mapping S to a two-level learner β_S, in the specific context of machine learning. Thus we have made connections between Rice's framework and HPO/"model selection". In the setting we described in this section, Rice's quest for a selection mapping S (i.e. the Algorithm Selection Problem) becomes in the machine learning framework of neural architecture search and meta-learning the quest for a learner β_S that one can apply to *any* learning task. More generally, finding an optimal β is the goal of meta-learning, the principal constituent of AutoML. Conversely, any β algorithm can be thought of as a β_S, with one trivial higher-level S returning always the same learner β. Thus, from now on we drop the index S of β_S.

3.1.5 . Meta-learning: Optimizing approximations of the performance model

Meta-learning aims to solve the Algorithm Selection Problem for machine learning (with the help of *meta-data*). This can be done by, for example, solving the optimization problem in Eq. (3.4), with the performance model $\mathfrak{p}$. Solving (3.4) is trivial when the performance model can be easily computed (just select the algorithm with the best performance). Hence, the difficulty lies in approximating the performance model $\mathfrak{p}$ as well as possible, which is the object of this section.

Typically, in machine learning, the performance model $\mathfrak{p}$ defined in (3.9)

$$\mathfrak{p}(T,\beta) = \int L(\beta(T)(x),y)\, dP(x,y) = R(\beta(T)) \tag{3.12}$$

cannot be directly computed analytically for multiple reasons (firstly, the distribution P is unknown; secondly, even when P is known, computing an expectation usually cannot be done in closed forms). However, examples drawn from P are given, which allows to compute empirical estimates.

In this context, the Algorithm Selection Problem solution in Eq. (3.4) is approximated by the selection mapping defined as:

$$S : T \mapsto \underset{\beta\in\mathscr{B}}{\arg\min}\, \hat{\mathfrak{p}}(T,\beta). \tag{3.13}$$

where $\hat{\mathfrak{p}}$ is an estimation of $\mathfrak{p}$. How to obtain such estimation $\hat{\mathfrak{p}}$ is a crucial problem in meta-learning.

Many works (Feurer et al., 2015; Hutter et al., 2011a; Jin et al., 2019; Sun-Hosoya et al., 2018a) proposed to *modeling* or *approximating* the performance model $\mathfrak{p}$ based on past experiences (i.e. **meta-data**) of trying different algorithms on different problems and recording the performances. Concretely, a set of past experience called a **meta-dataset**

$$\mathfrak{D}_{tr} = \{(T_j,\beta_j,R_j)|\, j\in J\} \tag{3.14}$$

is considered. Here J is an index set and R_j is an estimation of the true performance $\mathfrak{p}(T_j,\beta_j)$ and one can use, for example, the validation error on the task T_j.

We will call an algorithm solving the Algorithm Selection Problem (for machine learning) *given a meta-dataset* a **meta-learner** (or a γ**-level algorithm**), having the signature

$$\gamma : \mathfrak{D}_{tr} \mapsto \beta.$$

Here we reduced the problem of finding a selection mapping to that of finding a learner, as introduced previously in Section 3.1.4.

According to the exploitation of the information in $\mathfrak{D}_{tr}$, we categorize methods learning the performance model to 3 classes (Figure 3.1):

— **Zero-order meta-learning** methods, related to the notion of *learning from model evaluations* according to Vanschoren (2018), only make use of the performance values R_j and consider meta-datasets of the following form

$$\mathfrak{D}_{tr} = \{R_j | j \in J\}$$

or

$$\mathfrak{D}_{tr} = \{R_{i,j} | i \in I, j \in J\}$$

where i and j are indices of the task and the learner respectively;

— **First-order meta-learning** methods, related to the notion of *learning from task properties* according to Vanschoren (2018), additionally make use of (meta-)features of the tasks and/or those of the learners. For these methods, the meta-dataset takes the form

$$\mathfrak{D}_{tr} = \{(\Phi(T_j), \Psi(\beta_j), R_j) | j \in J\}$$

where Φ and Ψ are some (meta-)feature extractors for the tasks and the learners respectively. Methods using hyperparameters to encode learning algorithms and using meta-features of datasets fall into this category;

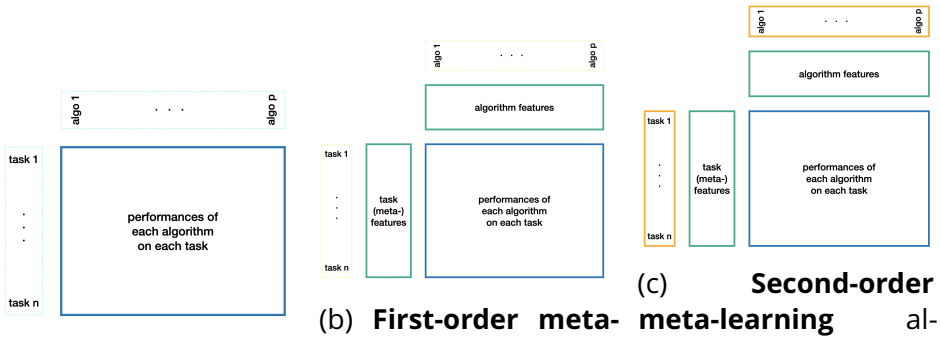
— **Second-order meta-learning** methods, related to the notion of *learning from prior models* according to Vanschoren (2018), use the full information with

$$\mathfrak{D}_{tr} = \{(T_j, \beta_j, R_j) | j \in J\}.$$

A contrast of these three classes of methods is visualized in Figure 3.1.

3.1.6 . Three-level formulation of algorithms

We now summarize the three levels (α, β, γ) of algorithms introduced before in Table 3.1. For ease of understanding, we associate each level to a method in the popular machine learning toolkit *scikit-learn* (Pedregosa et al., 2011).



(a) **Zero-order meta-learning** algorithms use only previously known performances of algorithms on tasks.

(b) **First-order meta-learning** algorithms use previously known performances plus meta-features of tasks and/or algorithms.

(c) **Second-order meta-learning** algorithms use full information of past tasks and algorithms (including all examples and all algorihtm code).

Figure 3.1 **Categorization of meta-learning in terms of the information used in the meta-dataset.**

This *three-level formulation* of algorithms serves as a footstep for this thesis. Many of our works are inspired or guided by this formulation. This includes the design of the AutoDL challenge (Chapter 4), the design of the MetaDL challenge (Chapter 6) and the theoretical analysis of zero-order meta-learning (Chapter 6). The reflection on GramNAS can also be retroactively absorbed by this three-level formulation (see Section 3.2.3).

3.2 . Meta-learning problems considered in this thesis

In this section, we position several main problems addressed in this thesis using the terminology we just defined.

3.2.1 . Algorithm selection in multi-phase challenges

In the AutoDL challenges that we will introduce in detail in Chapter 4, different participants' approaches (the learners) are evaluated on different datasets (the tasks). The leaderboard (of the first phase called the *feedback phase*) is thus a meta-dataset of the form

$$\mathfrak{D}_{tr} = \{R_{i,j} | i \in I, j \in J\} \tag{3.15}$$

Table 3.1 **Algorithms in the three-level formulation and the corresponding programmatic interface methods in *scikit-learn* and MetaDL challenge.**

Level	Type	Signature	Method name	Examples
α-level	predictor	$\alpha : x \mapsto y$	`predict` in *scikit-learn*	Trained or "hard-coded" image classifiers such as (Dalal and Triggs, 2005; Lowe, 2004)
β-level	learner	$\beta : T \mapsto \alpha$	`fit` in *scikit-learn*	Learner algorithms of SVM (Boser et al., 1992), NAS (Baker et al., 2016; Zoph and Le, 2016)
γ-level	meta-learner	$\gamma : \mathfrak{D}_{tr} \mapsto \beta$	`meta_fit` in MetaDL challenge	Meta-learners of MAML (Finn et al., 2017), ActivMetaL (Sun-Hosoya et al., 2018b)

where I indexes the set of tasks and J indexes the set of participants.

The job of the challenge organizers is thus to select the top participants/learners that can obtain good performances on unseen tasks in the second phase (called the *final phase*).

From the point of view of the challenge organizers, this forms a zero-order meta-learning problem.

3.2.2 . Active learning with dynamic meta-datasets

In the section 6.4 of Chapter 6, we will consider a problem setting where one can evaluate different learners on a *new* task and *update the meta-dataset* with these new evaluations.

Although the meta-dataset of this problem has similar form with the one in the previous section

$$\mathfrak{D}_{tr} = \{R_{i,j} | i \in I, j \in J\}, \tag{3.16}$$

we should keep in mind that for this problem, the meta-dataset $\mathfrak{D}_{tr}$ changes dynamically during the learning process. Obviously, this is a zero-order meta-learning problem.

Having a dynamic meta-dataset is very common in practice. This is the case for most NAS and HPO methods.

3.2.3 . NAS for one single task

The Neural Architecture Search (NAS) problem is a special form of Hyperparameter Optimization for Deep Learning. NAS composes an important research domain for Automated Deep Learning.

In NAS, one typically only considers *one single* task T_0 at a time (think of the CIFAR-10 (Krizhevsky, 2009) task for example). The corresponding meta-dataset thus has the form

$$\mathfrak{D}_{tr} = \{(T_0, \Psi(\beta_j), R_j) | j \in J\}. \tag{3.17}$$

Here the β_j's correspond to some learners defined by different neural network architectures and the Ψ can be, for example, a list of strings containing description of each layer in the network. Due to the meta-dataset form (3.17), NAS can be considered as first-order meta-learning.

We note that NAS approaches often have an aspect of active learning as introduced in the previous section. A NAS algorithm typically repeats the following iteration:

1. Recommend new promising learner (based on the meta-dataset $\mathfrak{D}_{tr}$) and evaluate it on T_0;
2. Record this evaluation and update the meta-dataset $\mathfrak{D}_{tr}$.

Although here we consider NAS as a special form of meta-learning, we still assign a separate chapter (Chapter 5) to NAS due to its specialty and importance.

3.2.4 . Few-shot learning

In Chapter 6, we will introduce the AAAI-2020 MetaDL challenge (Baz et al., 2021), which focuses on the *few-shot learning* problem in the meta-learning community.

In few-shot learning, the meta-dataset has the most general form

$$\mathfrak{D}_{tr} = \{(T_j, \beta_j, R_j) | j \in J\} \tag{3.18}$$

where the learners β_j are typically defined by a fixed neural network architecture *with different weight initialization*. The full information on the tasks T_j is also exploited since all examples of the tasks in $\mathfrak{D}_{tr}$ are used.

Few-shot learning is thus categorized as second-order meta-learning.

3.2.5 . Empirical Evaluation Setting

We now briefly introduce the datasets formatted and used during this thesis. These datasets constitute an important problem space $\mathscr{B}$ for this thesis.

In the AutoML community, datasets were essentially feature-based data, such as those in AutoML challenges (Guyon et al., 2015, 2018) or in benchmarking platforms such as OpenML (Vanschoren et al., 2014). However, feature-based data are not well suited for deep learning because learning directly from raw data (such as images or text) is a huge advantage of deep learning. So the data we use in this thesis follow, in most cases, a *4D-tensor format* described as follows. Each dataset is a set of example-label pairs. Each example is a 4D-tensor of shape $[t, H, W, C]$ with t for time axis, $\mathscr{H}$ and W for two space axes, and C for channel axis. Some of these four dimensions can be not fixed to an integer (think of documents or videos of different lengths). While for the labels, each is a vector of dimension K when there are K classes. Each of these K entries is 0 or 1 representing if the corresponding class is present in the given example.

This data format allows us to encode examples from many popular data domains such as image, video, speech, text and tabular. We note that multi-modal data such as videos with audio tracks or even with subtitles are not considered in our scope. How data from each domain are encoded is visualized in Figure 3.2.

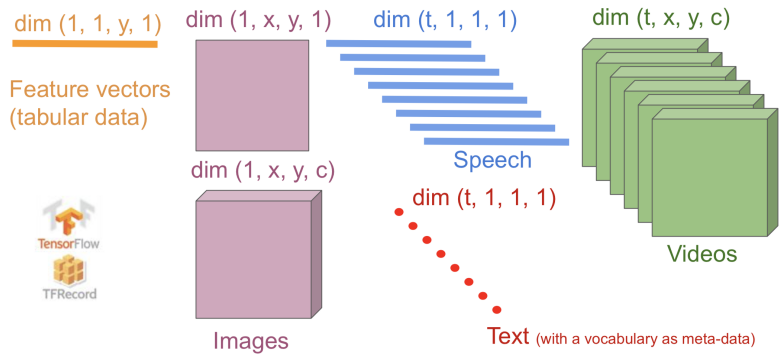


Figure 3.2 **4D-tensor format of the datasets we formatted during this thesis.** This format can encode data for which each example can be an image, a video, a document, an audio or a feature vector. All data are provided *raw* without pre-processing, in TensorFlow (Abadi et al., 2016) TFRecord format.

We list all datasets that are used in the AutoDL challenges (i.e. AutoCV, AutoCV2, AutoNLP, AutoSpeech and AutoDL) as well as in this thesis in Table 3.2. These datasets come from diverse domains. Some of them are used in more than two phases and/or challenges. We hope that this enriching repository of datasets can help the community to advance in the research of AutoML, especially meta-learning.

Table 3.2 **Datasets used in AutoDL challenges.** "HWR" means handwriting recognition, "chnl" channel, and "var" variable size.

#	Dataset	Challenge(s)	Phase	Domain	Type	Class num.	Sample number		Tensor dimension			
							train	test	time	row	col	chnl
1	Munster	AutoCV1	public	HWR	image	10	60000	10000	1	28	28	1
2	Chucky	AutoCV1	public	objects	image	100	48061	11939	1	32	32	3
3	Pedro	AutoCV1	public	people	image	26	80095	19905	1	var	var	3
4	Decal	AutoCV1	public	aerial	image	11	634	166	1	var	var	3
5	Hammer	AutoCV1	public	medical	image	7	8050	1965	1	600	450	3
6	Ukulele	AutoCV1	feedback	HWR	image	3	6979	1719	1	var	var	3
7	Caucase	AutoCV1	feedback	objects	image	257	24518	6089	1	var	var	3
8	Beatriz	AutoCV1	feedback	people	image	15	4406	1094	1	350	350	3
9	Saturn	AutoCV1	feedback	aerial	image	3	324000	81000	1	28	28	4
10	Hippocrate	AutoCV1	feedback	medical	image	2	175917	44108	1	96	96	3
11	Loukoum	AutoCV1 AutoCV2	final final	HWR	image	3	27938	6939	1	var	var	3
12	Tim	AutoCV1	final	objects	image	200	80000	20000	1	32	32	3
13	Apollon	AutoCV1 AutoCV2 AutoDL	final final feedback	people	image	100	6077	1514	1	var	var	3
14	Ideal	AutoCV1 AutoCV2	final feedback	aerial	image	45	25231	6269	1	256	256	3
15	Ray	AutoCV1 AutoDL	final final	medical	image	7	4492	1114	1	976	976	3
16	Kraut	AutoCV2	public	action	video	4	1528	863	var	120	160	1
17	Katze	AutoCV2	public	action	video	6	1528	863	var	120	160	1
18	Kreatur	AutoCV2	public	action	video	4	1528	863	var	60	80	1
19	Freddy	AutoCV2	feedback	HWR	image	2	546055	136371	1	var	var	3

20	Homer	AutoCV2	feedback	action	video	12	1354	353	var	var	var	3
21	Isaac2	AutoCV2	feedback	action	video	249	38372	9561	var	102	78	1
22	Formula	AutoCV2	feedback	misc.	video	4	32994	8203	var	80	80	3
23	Fiona	AutoCV2 AutoDL	final final	action	video	6	8038	1962	var	var	var	3
24	Monica1	AutoCV2 AutoDL	final feedback	action	video	20	10380	2565	var	168	168	3
25	Kitsune	AutoCV2	final	action	video	25	18602	4963	var	46	82	3
26	data01	AutoSpeech	public	speech	time	100	3000	3000	var	1	1	1
27	data02	AutoSpeech	public	speech	time	7	428	107	var	1	1	1
28	data03	AutoSpeech	public	speech	time	3	796	200	var	1	1	1
29	data04	AutoSpeech	public	speech	time	20	939	474	var	1	1	1
30	data05	AutoSpeech	public	speech	time	10	199	597	var	1	1	1
31	data11	AutoSpeech	feedback	speech	time	55	1300	2000	var	1	1	1
32	data12	AutoSpeech	feedback	speech	time	5	3120	346	var	1	1	1
33	data13	AutoSpeech	feedback	speech	time	3	5000	1330	var	1	1	1
34	data14	AutoSpeech	feedback	speech	time	8	767	191	var	1	1	1
35	data15	AutoSpeech	feedback	speech	time	76	2286	571	var	1	1	1
36	data21	AutoSpeech	final	speech	time	50	800	1200	var	1	1	1
37	data22	AutoSpeech	final	speech	time	4	2649	294	var	1	1	1
38	data23 Oreal	AutoSpeech AutoDL	final final	speech	time	3	2000	264	var	1	1	1
39	data24	AutoSpeech	final	speech	time	16	384	386	var	1	1	1
40	data25 Sahak	AutoSpeech AutoDL	final feedback	speech	time	100	3008	752	var	1	1	1
41	O1	AutoNLP	public	english	text	2	7792	1821	var	1	1	1
42	O2	AutoNLP	public	english	text	20	11314	7532	var	1	1	1
43	O3	AutoNLP	public	english	text	2	60000	40000	var	1	1	1
44	O4	AutoNLP	public	chinese	text	10	55000	10000	var	1	1	1
45	O5	AutoNLP	public	chinese	text	18	156000	72000	var	1	1	1
46	PU1	AutoNLP	feedback	english	text	9	2822	499	var	1	1	1
47	PU2	AutoNLP	feedback	english	text	5	132651	23409	var	1	1	1
48	PU3	AutoNLP	feedback	chinese	text	2	1110203	195919	var	1	1	1
49	PU4	AutoNLP	feedback	chinese	text	11	100000	50000	var	1	1	1
50	PU5	AutoNLP	feedback	chinese	text	31	600000	400000	var	1	1	1
51	PR1	AutoNLP	final	english	text	20	33807	5967	var	1	1	1
52	PR2 Tanak	AutoNLP AutoDL	final feedback	english	text	2	42500	7501	var	1	1	1
53	PR3	AutoNLP	final	english	text	4	90000	30000	var	1	1	1
54	PR4	AutoNLP	final	chinese	text	11	100000	50000	var	1	1	1
55	PR5 Tal	AutoNLP AutoDL	final final	chinese	text	15	250000	132688	var	1	1	1
56	Adult	AutoDL	public	categorical	tabular	5	39074	9768	1	1	24	1
57	Dilbert	AutoDL	public	objects	tabular	5	14860	9720	1	1	2000	1
58	Digits	AutoDL	public	HWR	tabular	10	35000	35000	1	1	1568	1
59	Madeline	AutoDL	public	artificial	tabular	2	4220	3240	1	1	259	1
60	Barak	AutoDL	feedback	CE pair	tabular	4	21869	2430	1	1	270	1
61	Bilal	AutoDL	final	audio	tabular	20	10931	2733	1	1	400	1
62	Cucumber	AutoDL	final	people	image	100	18366	4635	1	var	var	3
63	Yolo	AutoDL	final	action	video	1600	836	764	var	var	var	3
64	Marge	AutoDL	final	music	time	88	9301	4859	var	1	1	1
65	Viktor	AutoDL	final	english	text	4	2605324	289803	var	1	1	1
66	Carla	AutoDL	final	neural	tabular	20	10931	2733	1	1	535	1

We show a distribution of these datasets in Figure 3.3, using meta information of each dataset such as the number of examples and the compressed size. Each domain has a corresponding color.

3.3 . Contributions of the Author

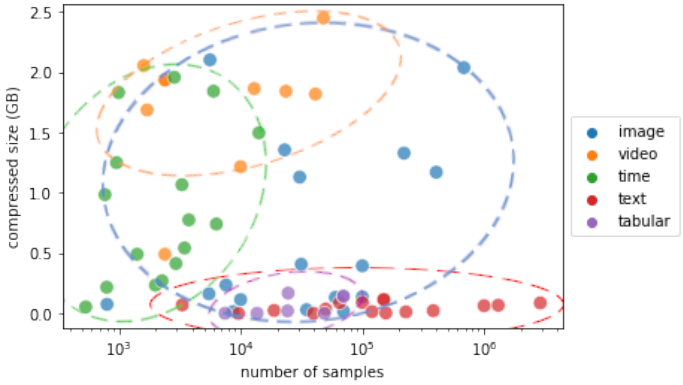


Figure 3.3 **Distribution of AutoDL challenge datasets** with respect to compressed storage size in giga-bytes and total number of examples for all 66 AutoDL datasets. We see that the text domain varies a lot in terms of number of examples but remains small in storage size. The image domain varies a lot in both directions. Video datasets are large in storage size in general, without surprise. Speech and time series datasets have fewer number of examples in general. Tabular datasets are concentrated and are small in storage size.

To end this chapter, we briefly summarize the major contributions of the author during this thesis, under the previously introduced scope.

— **Mathematical formulation and unified framework for AutoML.** To formulate the AutoML problem in a rigorous way, we introduce in Chapter 3 a mathematical framework that: (1) formulates all problems treated in this thesis as an Algorithm Selection Problem (Rice, 1976) and categorizes all involved algorithms into three levels (α, β and γ levels); (2) concretely defines the concept of a *task* (especially in a supervised learning setting); (3) formally relates HPO and meta-learning; (4) introduces in Chapter 4 an *any-time learning metric* that allows to evaluate learning algorithms by not only their accuracy but also their learning speed, which is crucial in settings such as hyperparameter optimization (including neural architecture search) or meta-learning. This mathematical framework unifies different sub-fields of ML (*e.g.* HPO, NAS and meta-learning), allows us to systematically classify methods, and provides us with formal tools to facilitate theoretical developments and empirical research. In particular,

it serves as the theoretical basis of a series of challenges that we organized;

— **Large-scale AutoDL benchmark.** Our principal methodological approach to tackle AutoML with Deep Learning has been to set up an extensive benchmark, in the context of a challenge series on Automated Deep Learning (AutoDL), co-organized with ChaLearn, Google, and 4Paradigm. These challenges provide a benchmark suite of baseline AutoML solutions with a repository of around 100 datasets (from all above domains), over half of which are released as public datasets to enable research on meta-learning. The author developed and maintained the data formatting toolkit, the code (i.e. ingestion program [2] and scoring program [3]) defining the logic behind the challenges, the code of several baselines and also the toolkit for post-challenge analysis. The challenge platform, the starting kit, the dataset formatting toolkit and all winning solutions are open-sourced at https://autodl.chalearn.org/;

— **Extensive post-challenge analyses and meta-study.** At the end of these challenges, we carried out extensive post-challenge analyses which revealed that: (1) winning solutions generalize to new unseen datasets, which validates progress towards universal AutoML solution; (2) Despite our effort to format all datasets uniformly to encourage generic solutions, the participants adopted specific workflows for each modality; (3) Any-time learning was addressed successfully, without sacrificing final performance; (4) Although some solutions improved over the provided baseline, it strongly influenced many; (5) Deep learning solutions dominated, but Neural Architecture Search was impractical within the time budget imposed. Most solutions relied on fixed-architecture pre-trained networks, with fine-tuning. Ablation studies revealed the importance of meta-learning, ensembling, and efficient data

2. What is an ingestion program: https://github.com/codalab/codalab-competitions/wiki/User_Building-an-Ingestion-Program-for-a-Competition

3. What is a scoring program: https://github.com/codalab/codalab-competitions/wiki/User_Building-a-Scoring-Program-for-a-Competition

loading, while data-augmentation is not critical. All code and data (including post-challenge analyses data) are available at autodl.chalearn.org. The post-challenge analysis is introduced in Chapter 4 and the corresponding paper (Liu et al., 2021) is published in IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI);

— **Development of custom AutoDL methods.** In Chapter 5, our proposed method *GramNAS* tackles the neural architecture search (NAS) problem by using a *formal grammar* to encode neural architectures. This provides a solution to algorithm representation and opens the possibility to analyze learning of algorithms from the essential: after all, the ultimate representation of an algorithm is *its code* (together with a compiler). Two alternative approaches have been experimentally investigated: one based on Monte-Carlo Tree Search (MCTS) and one based on an evolutionary algorithm. As tree structures arise very naturally with formal grammars, Monte-Carlo Tree Search may be used rather naturally as search algorithm. The MCTS GramNAS algorithm achieves near state-of-the-art performance (94% accuracy) on CIFAR-10 dataset. We also cast on our GramNAS framework the AgEBO (Aging Evolution with Bayesian Optimisation) algorithm to illustrate the other approach. This last algorithm lends itself to parallelism. In a benchmark on 4 large well-known datasets, it beats state-of-the-art packages AutoGluon and AutoPytorch. The GramNAS framework provides insights to the understanding and representation of learning algorithms. A tool-kit was open-sourced to craft customized formal grammars for novel applications, allowing users to reuse common underlying search strategies.

— **Laying the basis for a future challenge on meta-learning.** The AutoDL challenge series revealed the importance of meta-learning to succeed in solving AutoDL tasks. Yet the challenge setting did not evaluate meta-learning in the sense that meta-learning was not carried out on the challenge platform: code submitted by participants was only trained and tested indepen-

dently on several tasks. With an intern Adrian El Baz, we designed a new meta-learning challenge protocol and proposed the MetaDL challenge for evaluating few-shot learning approaches. The MetaDL challenge is accepted as an official competition of NeurIPS 2021 [4] and will terminate in October 2021. A repository, Meta-Album (Ullah et al., 2021), of 15 meta-datasets that aim to bridge the gap between small-scale benchmarks (such as Omniglot (Lake et al., 2015)) and large-scale benchmarks (such as Meta-dataset (Triantafillou et al., 2019)) is proposed and the corresponding paper is under review for NeurIPS 2021 Datasets and Benchmarks Track;

— **Theoretical contributions.** During the course of this thesis, several collaborations were entered to tackle problems of transfer learning and expressiveness of neural networks. In Chapter 6, we formulated meta-learning in a reinforcement learning setting and proved that under certain conditions, the average performance of the random search cannot be beaten (Liu and Guyon, 2021). Also in Chapter 6, we made theoretical analysis on the super-generalization ability of the LEAP nets proposed by us and proved that when the perturbations of the system are additive, LEAP nets are capable of super-generalization (Donnot et al., 2019; Donon et al., 2020b).

4. https://neurips.cc/Conferences/2021/CompetitionTrack

4 - AutoDL challenges

In this chapter, we present the design and results of a series of competitions in Automated Deep Learning (AutoDL). In this AutoDL challenge series 2019, we organized 5 machine learning challenges: AutoCV, AutoCV2, AutoNLP, AutoSpeech and AutoDL. If we use the terminology in Chapter 3, all of these challenges are in β-level (i.e. participants need to provide a β-level algorithm for submission). While for the organizers, they are facing a γ-level problem with the meta-dataset of the form

$$\mathfrak{D}_{tr} = \{R_{i,j} | i \in I, j \in J\}$$

where I indexes the tasks and J indexes the participants. The first 4 challenges concern each a specific application domain, namely computer vision, natural language processing and speech recognition respectively. And the last one AutoDL combines all domains together. Some highlights of this chapter include: (1) a benchmark suite of baseline AutoML solutions, with emphasis on domains for which Deep Learning methods have had prior success (image, video, text, speech, etc); (2) a novel "anytime learning" framework, which opens doors for further theoretical consideration; (3) a repository of around 100 datasets (from all above domains) over half of which are released as public datasets to enable research on meta-learning; (4) analyses revealing that winning solutions generalize to new unseen datasets, validating progress towards universal AutoML solution; (5) open-sourcing of the challenge platform, the starting kit, the dataset formatting toolkit, and all winning solutions. All information available at autodl.chalearn.org.

Organizing AutoDL challenges serves as an important link to other parts of this thesis and is of great scientific value in itself too. In the following, we list some major motivations on the organization of AutoDL challenges.

— Solving the AutoDL problem is scientifically very challenging and could be too hard for anyone to solve alone. Thus the community effort should be at help. Organizing challenges is a great

way to call for community efforts and also to standardize the problem. Besides learning a great deal of organizational skills and coordinating skills, organizing challenges allows the author to make one essential contribution: formulate the problem in a way that one can get conclusive results. By formulating the problem in a standard and mathematical, the author learned about the experimental design and came up with the definition of any-time learning;

— By organizing open and shared benchmarks and challenges, one can avoid the inventor-evaluator bias, which happens very often when one defines his/her problem and evaluate alone. Biases on the data or on the evaluation are often introduced unintentionally. With a community, the evaluation is more robust and fairer. In addition, I can put my own contributions together with others in some cases. This happened typically during the preparation of several baseline methods;

— Organizing challenges is an effective way to speed up getting results and accelerate research. The results come from many people and new ideas are more likely to occur since one's ideas constantly inspire the others'. Also, the fact that results come from many people allows us to carry out meta-study (paper accepted at TPAMI (Liu et al., 2021)). Such meta-studies allow us to provide a global view on the state of the community and help identify the bottlenecks of research to guide next steps;

— Formatting a large set of benchmark tasks allows us to pave the way of meta-learning research. Indeed, the work on dataset formatting provides a rich repository for the community to do research on meta-learning, which is of long-lasting scientific value.

4.1 . Introduction

Machine Learning (ML) keeps delivering impressive novel applications in our daily lives. But it is still facing enormous challenges, preventing its more universal deployment by users having direct needs but no time or resources to hire ML experts. In fact, even for ML experts,

effectively tuning hyper-parameters is still a daunting task, particularly for Deep Learning models, let alone addressing higher level aspects of model design, including problem definition, experimental design, data collection, pre-processing, design of metrics, computation of error bars, detection of bias in data, etc. Certainly, automating the entire modeling pipeline is still a far reaching goal, but the challenges we present in this paper allowed us to make great strides. We present here the results of the Automated Deep Learning (AutoDL) challenge series, addressing tasks in Computer Vision (Liu et al., 2019b), Natural Language Processing (NLP), Speech Recognition, etc. With the solutions provided (and open-sourced) by the winners, users must only preprocess data to horseshoe-fit them in a generic tensor format to have automated algorithms train and test Deep Learning neural networks for them. The problems addressed are multi-label classification, in an amazingly broad range of application domains (medical imaging, object or gesture classification, satellite imaging, to name a few). Besides saving human effort, the benefit of such automated solutions include reproducibility and accountability, freeing us potentially from the variability of human solutions and possibly increasing reliability.

The AutoDL challenge series is part of a larger effort on Automated Machine learning (AutoML) with **code submission** in which the solutions of participants are blind tested on the CodaLab challenge platform. In all of our AutoML challenges we seek to enforce learning within a fixed time budget and limited computational resources. One particularity of AutoDL challenges, compared to previous AutoML challenges, is that we seek to enforce **any-time learning**, which encourages solutions performing reasonably good early on in the whole learning process. This is achieved by using the Area under the Learning Curve (ALC) metric, as explained in Section 4.2.1. To help participants develop their code, we provide a starting kit in Python with TensorFlow/PyTorch interfaces, sample "public" datasets and sample code submissions. Some basic facts about the challenge series are summarized in Table 1.1.

While most of our challenges are run in two phases (a **feedback phase** with immediate feedback on a leaderboard on $N = 5$ practice datasets and a **final phase** with a single evaluation on $N = 5$ final test

datasets), in AutoCV, we evaluated the participants on the results of the feedback phase, to make it slightly easier. However, we ran privately a final test phase of which we report here the results. Since the 5 AutoCV final phase datasets were not disclosed, we re-used some in subsequent phases. AutoCV2 was run regularly in 2 phases. Even practice datasets during the feedback phase were not revealed to the participants (they were solely visible to their "autonomous agent").

4.2 . Data

In AutoDL challenges, *raw data* (images, videos, audio, text, etc) are provided to participants formatted in a uniform tensor manner (namely TFRecords, a standard generic data format used by TensorFlow). Statistics and meta-features of all AutoDL datasets are presented in Table 3.2.

For images with native compression formats (e.g. JPEG, BMP, GIF), we directly use the bytes. Our data reader decodes them on-the-fly to obtain a 4D tensor. Video files in mp4/avi format (without the audio track) are used in a similar manner. For text datasets, each example (i.e. a document) is a sequence of integer indices. Each index corresponds to a word (for English) or character (for Chinese) in a vocabulary given in the metadata. For speech datasets, each example is represented by a sequence of floating numbers specifying the amplitude at each timestamp, similar to uncompressed WAV format. Lastly, tabular datasets' feature vector representation can be naturally considered as a special case of our 4D tensor representation.

For practical reasons, each dataset was kept under 4GB, which required sometimes reducing image resolution, cropping, and/or downsampling videos. We made sure to include application domains in which the scales varied a lot. We formatted around 100 datasets in total and used 61 of them for AutoDL challenges: 16 image, 9 video, 15 text, 15 speech and 6 tabular. All datasets marked "public" can be downloaded on corresponding challenge websites, for example on the Get Data page of AutoDL challenge. All tasks are supervised multi-label classification problems, *i.e.* data samples are provided in pairs $\{X, Y\}$, X being an

input 4D tensor of shape (time, row, col, chnl) and Y a target binary vector (withheld from in test data).

4.2.1 . Evaluation Metrics

AutoDL challenges enforce *any-time learning* by scoring participants with the Area under the Learning Curve (ALC) (Figure 4.1).

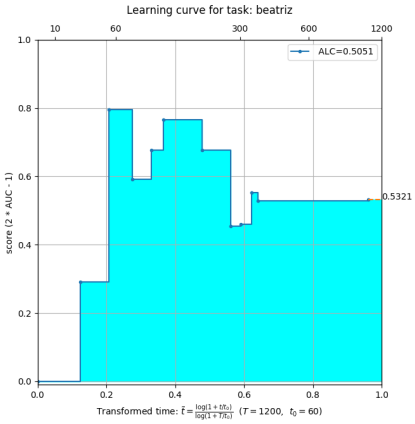


Figure 4.1 **Example of learning curve.** We modified the CodaLab competition platform (noa) so participants can save their results, at any intervals they choose, to incrementally improve their performance, until the time limit is attained. In this way, we can plot their learning curves: performance as a function of time. By evaluating them with the area under the learning curve, we push them to implement any-time learning methods. The x-axis corresponds to timestamp but normalized to [0,1]. This figure shows an example of possible over-fitting in which the participant could have stopped further training earlier.

The participants can train in increments of a chosen duration (not necessarily fixed) to progressively improve performance, until the time limit is attained. Several predictions can be made during the learning process and this allows us to plot their learning curves, i.e., "performance" as a function of time. More precisely, for each prediction made at a timestamp, we compute for each (binary) class the *Area Under ROC Curve* (AUC), then normalize it (and average over all classes) by

$$NAUC = 2 \times AUC - 1.$$

Then for each dataset, we compute the Area under Learning Curve (ALC) of a submission as follows:

— at each timestamp t, we compute $s(t)$, the $NAUC$ (see above) of the most recent prediction. In this way, $s(t)$ is a step function with respect to timestamp t;

— in order to normalize time to the $[0,1]$ interval, we perform a time transformation by

$$\tilde{t}(t) = \frac{\log(1+t/t_0)}{\log(1+T/t_0)}$$

where T is the time budget (e.g. 1200 seconds = 20 minutes) and t_0 is a reference time amount (e.g. 60 seconds).

— then compute the area under learning curve using the formula

$$\begin{aligned} ALC &= \int_0^1 s(t) d\tilde{t}(t) \\ &= \int_0^T s(t)\tilde{t}'(t) dt \\ &= \frac{1}{\log(1+T/t_0)} \int_0^T \frac{s(t)}{t+t_0} dt \end{aligned}$$

we see that $s(t)$ is weighted by $1/(t+t_0)$, giving a stronger importance to predictions made at the beginning of th learning curve.

The ALC gives the evaluation score for one task. Finally, when ALC score is computed for all tasks, the final score is obtained by the **average rank** (over all tasks among all submissions). It should be emphasized that multi-class classification metrics are not being considered, i.e., each class is scored independently.

4.3 . Baselines

As each challenge (except for AutoDL) involves a specific domain, different baselines are provided for different challenges.

4.3.1 . Baselines for AutoCV & AutoCV2

For AutoCV and AutoCV2, we introduced 3 baseline methods with varied complexity and computer resource requirements. **Baseline 0** makes one single all-zero prediction and always gets 0 NAUC score (hence 0 ALC as well). **Baseline 1** is a linear classifier. It uses a cross entropy loss and an Adam optimizer (Kingma and Ba, 2014). If the input shape is variable, resize all images to a fixed shape 112×112. When the number of frames (time axis) is variable, sample 10 consecutive frames at random, both for training and testing. The scheduling strategy is to double the number of training steps at each iteration. Stop training and predicting when time budget is not enough for next iteration. **Baseline 2** uses a neural network architecture determined by the tensor shape of the input examples. More concretely, 3D convolutional layer is repeatedly applied followed by a 3D max-pooling layer, until the number of neurons of the hidden layer is less than a pre-defined number (e.g. 1000), then apply a fully connected layer for classification. More details on these baselines can be found in Liu et al. (2020a). Lastly, we also prepared two private baselines with fixed backbone architecture ResNet-50 (He et al., 2015) and Inception-V3 (Szegedy et al., 2016) but these baselines are only used by the organizers for testing and comparison purpose.

4.3.2 . Baselines for AutoNLP

For AutoNLP, **Baseline 0** uses a Support Vector Machine (SVM) as classifier. The input text is preprocessed by keeping only the alphanumeric characters and been vectorized with the Term Frequency Inverse Document Frequency (TF-IDF) vectorizer with a maximum vocabulary size of 20000.

Baseline 1 follows and uses a convolutional neural network (CNN). In this method, the text input data is preprocessed like in the previous baseline. The vocabulary is indexed and the integer sequence is then padded with a maximum sequence length. The model architecture consists of an Embedding layer with a dimension of 200, two 1D convolutional layers, a Maxpooling layer, two 1D convolutional layers, a Global Average pooling layer and a fully connected layer.

Compared to Baseline 1, **Baseline 2** uses pre-trained word embedding weights in addition. This method uses the same pre-processing,

vectorization and model architecture from the previous baseline with an embedding layer of dimension 300. The embedding layer weights are loaded with a pre-trained embedding from FastText (Joulin et al., 2017).

4.3.3 . Baseline for AutoSpeech

The baseline method for AutoSpeech is relatively straightforward. Features are extracted on each dataset using Mel-Frequency Cepstral Coefficients (MFCC) (Mermelstein, 1976), with shape padding. We then apply a CNN backbone model on the extracted preprocessed features, automatically adapting the number of layers according to the number of features. We train only one iteration or perform early stopping at convergence. For prediction, the same MFCC feature pre-processing is applied and use the trained model for inference.

4.3.4 . Baseline for AutoDL

For the final AutoDL challenge, we provide a baseline referred to as **Baseline 3**, which is a combination of the winner solutions of AutoCV (*kakaobrain*), AutoNLP (*upwind_flys*) and AutoSpeech (*PASA_NJU*), using domain inference which depends only on the input shape of the 4D tensor. On tabular datasets (which are never used in above challenges), the model chosen is simply a fully connected neural network with 2 hidden layers of 256 neurons.

4.4 . Challenge results for AutoCV, AutoCV2, AutoNLP and AutoSpeech

In this section, we present the results and analysis of AutoCV, AutoCV2, AutoNLP and AutoSpeech. As AutoDL is a combination of domains for these challenges, its results will be presented in a separated section.

We only consider the **top-10 participants in the final phase** of each challenge for all analyses. The names of the top-3 teams can be found in Table 4.1.

4.4.1 . Learning curves obtained in each challenge

Table 4.1 **Top-3 winners** and **Pearson correlation coefficient** between average ranking vectors in feedback phase and final phase, with corresponding p-value. For all challenges except AutoCV2, the Pearson correlation is close to 1 with significant p-value, which means that the feedback phase results and final phase results are consistent, suggesting the generalization ability of these AutoDL methods. For AutoCV2, top-10 participants used very similar approaches (all similar to the solution of *kakaobrain*, in AutoCV), which makes the performances of different teams very close.

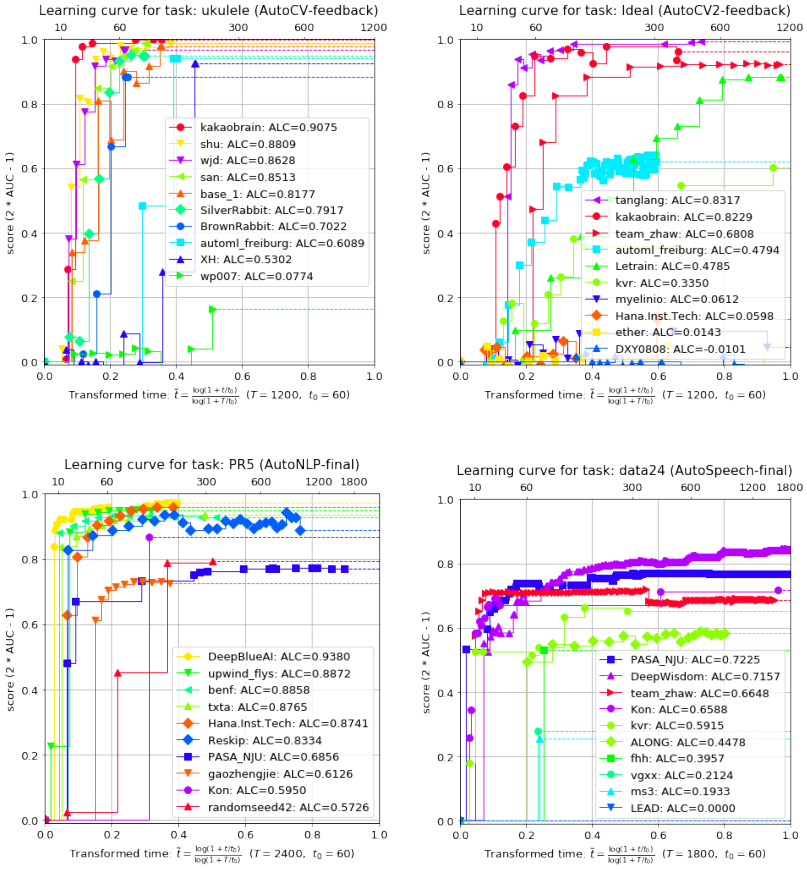
Challenge	Top-3 teams	Pearson's r	p-value
AutoCV	*kakaobrain*, *DKKimHCLee*, *base_1*	0.8321	2.836×10^{-3}
AutoCV2	*kakaobrain*, *tanglang*, *kvr*	0.3555	3.133×10^{-1}
AutoNLP	*DeepBlueAI*, *upwind_flys*, *txta*	0.8718	1.010×10^{-3}
AutoSpeech	*PASA_NJU*, *DeepWisdom*, *Kon*	0.8761	8.844×10^{-4}
AutoDL	*DeepWisdom*, *DeepBlueAI*, *PASA_NJU* and *Inspur_AutoDL* (tied)	0.9106	5.843×10^{-4}

For a given task, we plot all learning curves of top-10 participants in the same figure, for a clear comparison. In Figure 6.17, four of such figures are shown, each from a different challenge. From these curves, one can spot very different learning curve patterns, suggesting very different learning and predicting strategies.

4.4.2 . Generalization ability of AutoML methods

To evaluate the generalization ability of AutoML methods on unseen datasets, we compute the average rank over 5 datasets in both phases and consider their correlation (recall that datasets are DIFFERENT in each phase). This gives an average rank pair $(r_1, r_2) \in \mathbb{R}^2$ for each participant. When r_1 is close to r_2, the method is considered to super-generalize,*i.e.* generalize in the AutoML sense to NEW datasets, not just to a different test set from the same datasets as in common ML challenges. We plot all these pairs (r_1, r_2) in Figure 4.3. We observe that most participants' rank pairs are close to the diagonal, suggesting generalization ability for most methods. Some outliers such as *LEAD* are due to technical failure of code execution, e.g. with an out of memory

Figure 4.2 **Learning curves** for one specific task in each challenge. From these curves, we can already see that the strategies used by participants vary a lot. The number of predictions (i.e. number of points on a learning curve) ranges from 1 (e.g. in (c), *Kon* on *PR5* in AutoNLP final phase) to 789 of *DeepWisdom* on *data24* in AutoSpeech final phase, in (d). And from whether the curve decreases dramatically at some point (e.g. in (a), *base_1* and *XH* on *ukulele*), we can infer whether the submitted method uses a validation set to determine if a prediction should be made.



(OOM) error. And to evaluate the soundness of the choice of datasets for evaluating generalization, we also compute the Pearson correlation coefficient for all r_1, r_2 in Table 4.1 using $\rho_{X,Y} = \mathrm{cov}(X,Y)/(\sigma_X \sigma_Y)$.

4.4.3 . Modeling difficulty of datasets

To benchmark the modeling difficulty of each task/dataset, Figure 4.4 shows the best-vs-worst performances among top-10 participants.

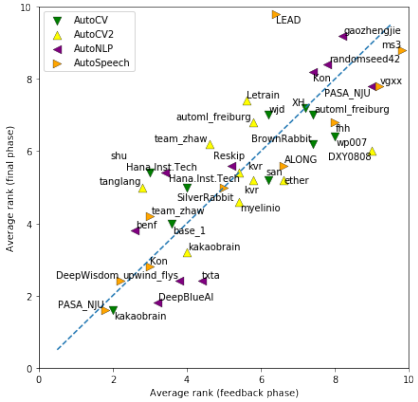


Figure 4.3 **Generalization ability** of AutoML methods. For each participant in a challenge, the average rank (over 5 datasets) in both phases is computed as x-axis and y-axis. When the scattered point is close to the diagonal, the feedback phase (with leaderboard feedback) result and final phase (with unseen datasets) result are consistent.

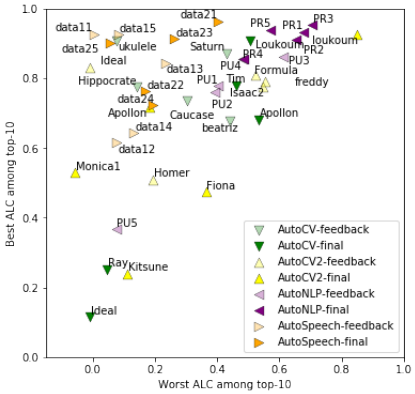


Figure 4.4 **Modeling difficulty** of each task/dataset. The x-axis (resp. y-axis) is the minimum (resp. maximum) ALC among top-10 participants in each challenge-phase. Tasks on top-left have larger modeling difficulty, while those close to the diagonal have small performance variance and model difficulty.

We see that many datasets from AutoNLP such as *PR1* to *PR5* are found on the top-right, meaning that the performance variance is small. This could be due to the pre-trained word embedding weights from FastText (Joulin et al., 2017) or BERT (Devlin et al., 2018) we provide in a Docker image shared by all participants. With these pre-trained word embedding, even weak classifiers could obtain relatively good performance. On the other hand, datasets from AutoSpeech such as *data11* and *data25* have large modeling difficulty, which might be related to the fact that raw speech datasets often require careful pre-processing steps in order to train a successful classifier.

4.4.4 . Addressing the any-time learning problem

The Figure 4.5 informs on participant's effectiveness to address the *any-time learning* problem. We first factored out dataset difficulty by re-scaling ALC and NAUC scores (resulting scores on each dataset having

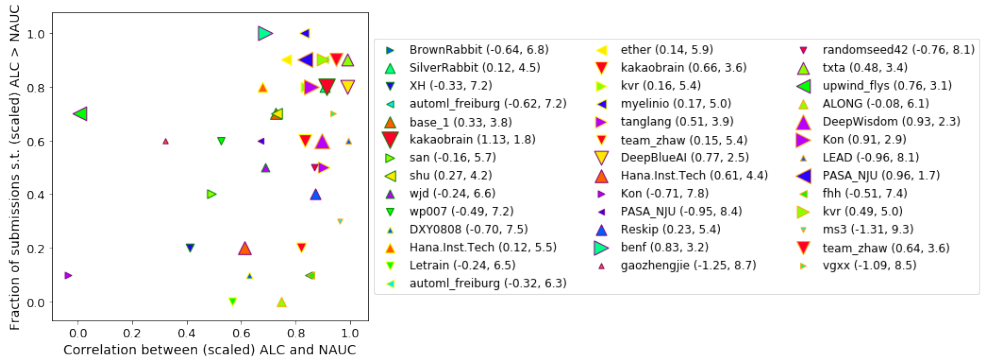


Figure 4.5 CORR vs FRAC (%(ALC $>$ NAUC) *vs* correlation(ALC, NAUC)). ALC and NAUC were "scaled" (see text). The numbers in the legend are average scaled ALC and average rank of each participant. The marker size increases monotonically with average scaled ALC. 34 out of 40 participants have a CORR greater than 0.5 and 30 out of 40 participants have a FRAC above 0.5.

mean 0 and variance 1). Then we plotted, for each participant, their fraction of submissions in which ALC is larger than NAUC (FRAC for short) *vs.* $correlation(ALC, NAUC)$ (CORR for short).

The participants in the bottom half of the figure did not address well the *any-time learning* problem because their FRAC is lower than 50%. Those participants did not perform well in the challenge either (small symbols). The participants that did well in the challenge (large symbols) are all in the upper right quadrant, with both FRAC larger than 50% and CORR larger than 0.7.

4.5 . Challenge results for AutoDL

The AutoDL challenge (the last challenge in the AutoDL challenges series 2019) lasted from 14 Dec 2019 (launched during NeurIPS 2019) to 3 Apr 2020. It has had a participation of 54 teams with 247 submissions in total and 2614 dataset-wise submissions. Among these teams, 19 of them managed to get a better performance (i.e. average rank over the 5 feedback phase datasets) than that of Baseline 3 in feedback phase and entered the final phase of blind test. According to our challenge rules, only teams that provided a description of their approach (by filling out some fact sheets we sent out) were eligible for getting a ranking

in the final phase. We received 8 copies of these fact sheets and thus only these 8 teams were ranked. These teams are (alphabetical order): *DeepBlueAI*, *DeepWisdom*, *frozenmad*, *Inspur_AutoDL*, *Kon*, *PASA_NJU*, *surromind*, *team_zhaw*. One team (*automl_freiburg*) made a late submission and isn't eligible for prizes but will be included in the post-analysis for scientific purpose.

The final ranking is computed from the performances on the 10 unseen datasets in the final phase. To reduce the variance from diverse factors such as randomness in the submission code and randomness of the execution environment (which makes the exact ALC scores very hard to reproduce since the wall-time is hard to control exactly), we re-run every submission several times and average the ALC scores. The average ALC scores obtained by each team is shown in Figure 4.6 (the teams are ordered by their final ranking). From this figure, we see that some entries failed constantly on some datasets such as *frozenmad* on *Yolo*, *Kon* on *Marge* and *PASA_NJU* on *Viktor*, due to issues in their code (e.g. bad prediction shape or out of memory error). On the other hand, some entries crashed only sometimes on certain datasets, such as *Inspur_AutoDL* on *Tal*, whose cause is related to some pre-processing procedure on text datasets concerning stop words. Otherwise, the error bars show that the performances of most runs are statistically consistent.

4.6 . Winning approaches

In this section, we present in detail the winning solutions from top-3 winning teams (*DeepWisdom*, *DeepBlueAI* and *PASA_NJU*) and the team *automl_freiburg* which made a late submission in feedback phase but ranked 5th in final phase. We considered interesting to introduce *automl_freiburg*'s approach due to their contributions and for scientific purpose.

A summary of the winning approaches on each domain can be found in Table 4.2. Another summary using a categorization by machine learning techniques can be found in Table 4.3. We see in Table 4.2 that almost all approaches used 5 different methods from 5 domains.

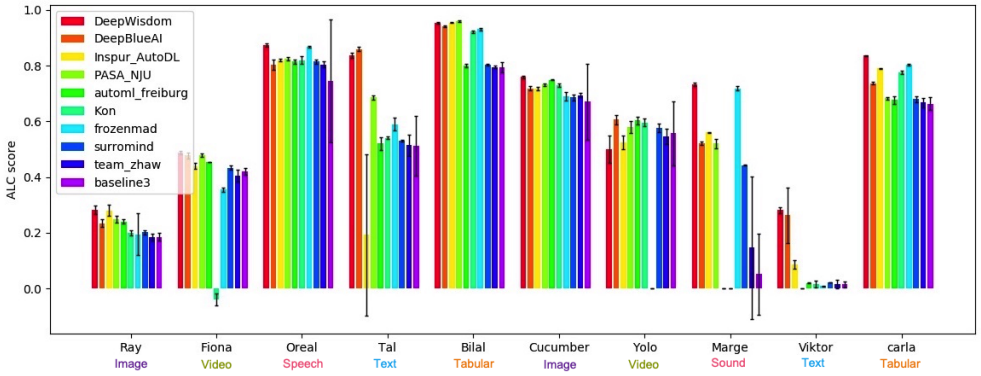


Figure 4.6 **ALC scores of top 9 teams in AutoDL final phase** averaged over repeated evaluations (and Baseline 3, for comparison). The entry of top 6 teams are re-run 9 times and 3 times for other teams. Error bars are shown with (half) length corresponding to the standard deviation from these runs. Some (very rare) entries are excluded for computing these statistics due to failures caused by the challenge platform back-end. The team ordering follows that of their average rank in the final phase. The domains of the 10 tasks are image, video, speech/times series, text, tabular (and then another cycle in this order). More information on the task can be found in Table 3.2.

For each domain, the winning teams' approaches are much inspired by Baseline 3. In Table 4.3, we see that almost all different machine learning techniques are actively present and frequently used in all domains (exception some rare cases for example transfer learning on tabular data). We'll introduce below in detail the top-3 winning solutions.

Table 4.2 **Summary of the five top ranking solutions** and their average rank in the final phase. The participant's average rank (over all tasks) in the final phase is shown in parenthesis (automl_freibug and Baseline 3 were not ranked in the challenge). Each entry concerns the algorithm used for each domain and is of the form "[pre-processing / data augmentation]-[transfer learning/meta-learning]-[model/architecture]-[optimizer]" (when applicable).

Team	image	video	speech	text	tabular
1.DeepWisdom (1.8)	[ResNet-18 and ResNet-9 models] [pre-trained on ImageNet]	[MC3 model] [pre-trained on Kinetics]	[fewshot learning] [LR, Thin ResNet34 models] [pre-trained on VoxCeleb2]	[fewshot learning] [task difficulty and similarity evaluation for model selection] [SVM, TextCNN,[fewshot learning] RCNN, GRU, GRU with Attention]	[LightGBM, Xgboost, Catboost, DNN models] [no pre-trained]
2.DeepBlueAI (3.5)	[data augmentation with Fast AutoAugment] [ResNet-18 model]	[subsampling keeping 1/6 frames] [Fusion of 2 best models]	[iterative data loader (7, 28, 66, 90%)] [MFCC and Mel Spectrogram pre-processing] [LR, CNN, CNN+GRU models]	[Samples truncation and meaningless words filtering] [Fasttext, TextCNN, BiGRU models] [Ensemble with restrictive linear model]	[3 lightGBM models] [Ensemble with Bagging]
3.Inspur_AutoDL (4)	Tuned version of Baseline 3				[Incremental data loading and training][HyperOpt][LightGBM]
4.PASA_NJU (4.1)	[shape standardization and image flip (data augmentation)][ResNet-18 and SeResnext50]	[shape standardization and image flip (data augmentation)][ResNet-18 and SeResnext50]	[data truncation(2.5s to 22.5s)][LSTM, VggVox ResNet with pre-trained weights of DeepWisdom(AutoSpeech2019) Thin-ResNet34]	[data truncation(300 to 1600 words)][TF-IDF and word embedding]	[iterative data loading] [Non Neural Nets models] [models complexity increasing over time] [Bayesian Optimization of hyperparameters]
5.frozenmad (5)	[images resized under 128x128] [progressive data loading increasing over time and epochs] [ResNet-18 model] [pre-trained on ImageNet]	[Successive frames difference as input of the model] [pre-trained ResNet-18 with RNN models]	[progressive data loading in 3 steps 0.01, 0.4, 0.7] [time length adjustment with repeating and clipping] [STFT and Mel Spectrogram pre-processing] [LR, LightGBM, VggVox models]	[TF-IDF and BERT tokenizers] [SVM, RandomForest , CNN, tinyBERT]	[progressive data loading] [no pre-processing] [Vanilla Decision Tree, RandomForest, Gradient Boosting models applied sequentially over time]
automl_freiburg	Architecture and hyperparameters learned offline on meta-training tasks with BOHB. Transfer-learning on unseen meta-test tasks with AutoFolio. Models: EfficientNet [pre-trained on ImageNet with AdvProp], ResNet-18 [KakaoBrain weights], SVM, Random Forest, Logistic Regression		Baseline 3		
Baseline 3	[Data augmentation with Fast AutoAugment, adaptive input size][Pretrained on ImageNet][ResNet-18(selected offline)]	[Data augmentation with Fast AutoAugment, adaptive input size, sample first few frames, apply stem CNN to reduce to 3 channels][Pretrained on ImageNet][ResNet-18(selected offline)]	[MFCC/STFT feature][LR, LightGBM, Thin-ResNet-34, VggVox, LSTM]	[resampling training examples][LinearSVC, LSTM, BERT]	[interpolate missing value][MLP of four hidden layers]

Table 4.3 **Machine learning techniques** applied to each of the 5 domains considered in AutoDL challenge.

ML technique	image	video	speech	text	tabular
Meta-learning	Offline meta-training transferred with AutoFolio Lindauer et al. (2015) based on meta-features (*automl_freiburg*, for image and video) Offline meta-training generating solution agents, searching for optimal sub-operators in predefined sub-spaces, based on dataset meta-data. (*DeepWisdom*) MAML-like method Finn et al. (2017) (*team_zhaw*)				
Preprocessing	image cropping and data augmentation (*PASANJU*), fast autoaugment (*DeepBlueAI*)	Sub-sampling keeping 1/6 frames and adaptive image size (*DeepBlueAI*) Adaptive image size	MFCC, Mel Spectrogram, STFT	root features extractions with stemmer, meaningless words filtering (*DeepBlueAI*)	Numerical and Categorical data detection and encoding
Hyperparameter Optimization	Offline with BOHB Falkner et al. (Bayesian Optimization and Multi-armed Bandit) (*automl_freiburg*) Sequential Model-Based Optimization for General Algorithm Configuration (SMAC) Hutter et al. (2011b) (*automl_freiburg*)		Online model complexity adaptation (*PASA_NJU*)	Online model selection and early stopping using validation set (*Baseline 3*(*upwind_flys*)	Bayesian Optimization (*PASANJU*) HyperOpt Bergstra et al. (2011) (*Inspur_AutoDL*)
Transfer learning	Pre-trained on ImageNet Russakovsky et al. (2015) (all teams except *Kon*)	Pre-trained on ImageNet Russakovsky et al. (2015) (all top-8 teams except *Kon*) MC3 model pre-trained on Kinetics (*DeepWisdom*)	ThinResnet34 pre-trained on VoxCeleb2 (*DeepWisdom*)	BERT-like Devlin et al. (2018) models pre-trained on FastText	(not applicable)
Ensemble learning	Adaptive Ensemble Learning (ensemble latest 2 to 5 predictions) (*DeepBlueAI*)	Ensemble Selection Caruana et al. (2004) (top 5 validation predictions are fused) (*DeepBlueAI*); Ensemble models sampling 3, 10, 12 frames (*DeepBlueA*)	last best predictions ensemble strategy (*DeepWisdom*) averaging 5 best overall and best of each model: LR, CNN, CNN+GRU (*DeepBlueA*)	Weighted Ensemble over 20 best models Caruana et al. (2004) (*DeepWisdom*)	LightGBM ensemble with bagging method Ke et al. (2017) (*DeepBlueAI*), Stacking and blending (*DeepWisdom*)

4.6.1 . Approach of *DeepWisdom* (1st prize)

The team *DeepWisdom* proposed a unified learning framework following a meta-learning paradigm. The framework consists of two parts: meta-train and meta-inference. The meta-train module takes as input the "public" datasets, which are augmented by the internal data augmentation engine, and the objective function (the ALC metric in the case of the challenge). The meta-trainer generates *solution agents*, whose objective is to search for best models, using search operators. In the meta-inference step, a new task is processed taking in one dataset of the challenge. Initial metadata and seed data (few-shot samples) are acquired from the raw dataset.This constitutes the input of the *solution agents* obtained by meta-training. Solution workflow starts after taking in the seed input data, then it receives more raw data in a streaming way, and interacts with a whole set of tables for storage to cache intermediate results and models. Next, we explain the domain-specific contributions of this team.

In the image domain, ResNet-18 is used in the early stages of the training and then switched to ResNet-9 in more advanced stages (The reason is the instability of ResNet-18). When switching from ResNet-18 to ResNet-9, to reduce I/O cost, they cache the mini batches, which have been used for ResNet-18 training in GPU and reuse them for the initial training phase of ResNet-9, until all these mini batches are exhausted. The networks are fine-tuned by initialing from Imagenet pre-trained networks. However, for a fast transfer learning batch normalization and bias variables are initialized from scratch. To avoid overfitting, fast auto augmentation is used in the later training phase, which can automatically search for the best augmentation strategy on the given dataset, according to the validation AUC. The searching process is quite time-consuming but effectively increase the top-AUC.

In the video domain, a mixed convolution (MC3) network Tran et al. (2018) is adopted which consists of 3D convolutions in the early layers and 2D convolutions in the top layers of the network. The network is pre-trained on the Kinetics dataset and accelerated transferring to other datasets by re-initializing linear weights and bias and freezing the first two layers. Due to the slower speed of 3D than 2D convolution,

3 frames are extracted at the early phase. Then for longer videos, an ensemble strategy is applied to combine best predictions from MC3 with 3-,10- and 12-frames data.

In the speech domain, a model search is applied in the meta-training part and LogisticRegression and ThinResnet34 achieve best performance in non-neural and neural models, respectively. The meta-trainer firstly learned that validating in the beginning was wasting the time budget without any effect on ALC, thus the evaluation agent did not validate when model was fitting new streaming data. Secondly, if amount of training samples was not very large, evaluation metric on training data could avoid overfitting partly while last best predictions ensemble strategy was applied.

In the text domain, they decode maximum 5500 samples for each round. Various data pre-processing methods are applied, including email data structure pre-processing, word frequency filtering and word segmentation. After tokenization and sequence padding, both pre-trained and randomly initialized word embedding (with various dimensions) are used as word features. The meta-trainer includes several solutions such as TextCNN, RCNN, GRU, and GRU with attention Kim (2014), Girshick et al. (2014). Hyperparameters are set after a neural network architecture is selected. Also a weighted ensembling is adopted among top 20 models based AUC scores.

Finally, in the tabular domain, they batch the dataset and convert tfdatasets to numpy format progressively, a weighted ensembling is applied based on several optimized models including LightGBM, Catboost, Xgboost and DNN on the offline datasets. To do so, data is split to several folds. Each fold has a training set and two validation sets. One validation set is used to optimize model hyperparameters and other set to compute ensembling weights.

4.6.2 . Approach of *DeepBlueAI* (2nd prize)

The DeepBlueAI solution is a combination of methods that are specific to each modality. Nevertheless, three concepts are applied across all modalities: 1) optimizing time budget by reducing the time for data processing, start with light models and parameters setting to acceler-

ate first predictions; 2) dataset adaptive strategies and 3) ensemble learning.

For images, the DeepBlueAI team applies a strategy adapted to each specific dataset. They apply a pre-trained ResNet-18 model. The dataset adaptive strategy is not applied to model selection but to parameters settings including: image size, steps per epoch, epoch after which starting validating and fusing results. With the aim to optimize for final AUC, and make results more stable, they apply a progressive ensemble learning method, i.e. for epochs between 5 to 10, the latest 2 predictions are averaged, while after 10 epochs the 5 latest predictions are averaged. When the score on validation set improves a little, data augmentation strategy is adopted by searching for the most suitable data augmentation strategy for each image dataset with a small scale version of Fast AutoAugment Lim et al. (2019a) limiting the search among 20 iterations in order to preserve more time for training.

For video, ResNet-18 is used for classification. In the search for a good trade-off between calculation speed and classification accuracy, 1/6 of the frames with respect to the total number are selected. For datasets with a large number of categories, image size is increased to 128 to get more details out of it. During training, when the score of the validation set increases, predictions are made on the test set, and submitted as the average of the current highest 5 test results.

For speech, features are extracted with Mel spectrogram Bridle and Brown (1974) for Logistic Regression (LR) model and MFCC Davis and Mermelstein (1980) for deep learning models. In order to accelerate the extraction long sequences are truncated but covering at least 90% of the sequence. Then, to accelerate first score computation, training data are loaded progressively, 7% for the first iteration, then 28%, 66% and then all data at 4th iteration, with care to balance multiple categories, to ensure the models can learn accurately. As for the models, LR is used for the first 3 iterations, then from the 4th iteration using all the data deep learning models, CNN and CNN+GRU Cho et al. (2014) are employed. At the end, the overall 5 best models and the best version of each of the 3 models are averaged to build a final ensemble. The iterative data loading is especially effective on large dataset and play

a significant role in the performance measured by the metric derived from the ALC.

For text, the data set size, text length and other characteristics are automatically obtained, and then a pre-processing method suitable for the data set is adopted. Long texts, over 6000 words are truncated, and NLTK stemmer is used to extract root features and filter meaningless words with frequency below 3. As for model selection, FastText Joulin et al. (2017), TextCNN Kim (2014), BiGRU Cho et al. (2014) are used by their system that generate different model structures and set of parameters adapted to each dataset. The size of the data set, the number of categories, the length of the text, and whether the categories are balanced are considered to generate the most suitable models and parameter settings.

For tabular, three directions are optimized: accelerating scoring time, adaptive parameter setting, ensemble learning.

Data is first split into many batches to significantly accelerate the data loading and converted from TFrecords to numpy format. In terms of models, decision trees LightGBM are adopted to get faster scoring than with deep learning models. Because LightGBM supports continuous training, and the model learns faster in the early stage. During the training phase, earnings from the previous epochs are much higher than those from the latter. Therefore, a complete training is intelligently divided into multiple parts. The result is submitted after each part to get score faster.

In terms of adaptive parameter setting, some parameters are automatically set according to the size of data and the number of features of the tables. If the number of samples is relatively large, the ensemble fraction is reduced. If the original features of the sample are relatively large, the feature fraction is reduced. A learning rate decay is applied, starting with a large value to ensure a speed up in the early training. An automatic test frequency is adopted. Specifically, the frequency of testing is controlled based on training speed and testing speed. If the training is slow and the prediction is fast, the frequency of the test is increased. On the contrary, if training is fast and prediction is slow, the frequency is reduced. This strategy can improve to higher early scores.

In order to improve generalization, multiple lightGBM models are used to make an ensemble with a bagging method.

4.6.3 . Approach of *PASA_NJU* (3rd prize)

The *PASA_NJU* team modeled the problem as three different tasks: CV (image and video), Sequence (speech and text) and Tabular (tabular domain).

For the CV task, they preprocessed the data by analysing few sample instances of each dataset at training stage (such as image size, number of classes, video length, etc) in order to standardize the input shape of their model. Then, simple transformations (image flip) were used to augment the data. Random frames were obtained from video files and treated as image database. For both Image and Video tasks, ResNet-18 He et al. (2015) is used. However, SeResnext50 Hu et al. (2018) was used at later stages. Basically, they monitor the accuracy obtained by the ResNet-18 model and change the model to the SeResnext50 if no significant improvement is observed.

Speech and Text data are treated similarly, i.e., as a Sequence task. In a pre-processing stage, data samples are cut to have the same shape. Their strategy was to increase the data length as time pass. For example, they use raw data from 2.5s to 22.5s in speech task, and from 300 to 1600 words when Text data is considered. In both cases, handcrafted feature extraction methods are employed. For speech data, mel spectrogram, MFCC and STFT Lowe (2004) is used. When Text is considered, TF-IDF and word embedding is used. To model the problem, they employed Logistic Regression at the first stages and use more advanced Neural Networks at later stages, such as LSTM and Vggvox Resnet Chung et al. (2018) (for speech data), without any hyperparameter optimization method. In the case of Vggvox Resnet, pre-trained model from *Deepwisdom*'s team from AutoSpeech Challenge 2019 Liu et al. (2020b) was used.

For Tabular data, they divided the entire process into three stages based on the given time budget, named Retrieve, Feature, and Model, and employed different models and data pre-processing methods at each stage, aiming to have quick responses at early stages. The main task of the Retrieve stage is to get the data and predict as soon as

possible. Each time a certain amount of data is acquired, a model is trained using all the acquired data. Thus, the complexity of the model is designed to increase with time. The main task of the Feature stage is to search for good features. As the Neural Feature Seacher(NFS) Chen et al. (2019) method uses RNN as the controller to generate the feature sequence, they used the same method and speed up the process by parallelizing it. Finally, at the Model stage, the goal is to search for a good model and hyperparameters. For this, they use hyperopt Bergstra et al. (2013), which is an open-source package that uses Bayesian optimization to guide the search of hyperparameters.

4.6.4 . Approach of *automl_freiburg*

In contrast to other teams, *automl_freiburg* adopts a domain-independent approach but focused only the computer vision tasks (i.e. image and video datasets) of this challenge. While for all other tasks *automl_freiburg* simply submitted the baseline to obtain the baseline results, they achieved significant improvement on the computer vision tasks w.r.t. the baseline method. To improve both efficiency and flexibility of the approach, they first exposed relevant hyperparameters of the previous AutoCV/AutoCV2 winner code Brain (2019) and identified well-performing hyperparameter configurations on various datasets through hyperparameter optimization with BOHB Falkner et al.. They then trained a cost-sensitive meta-model Xu et al. (2012) with AutoFolio Lindauer et al. (2015) – performing hyperparameter optimization for the meta-learner – that allows to automatically and efficiently select a hyperparameter configuration for a given task based on dataset meta-features. The proposed approach on the CV task is detailed next.

First, they exposed important hyperparameters of the AutoCV/AutoCV2 winner's code Brain (2019) such as the learning rate, weight decay or batch sizes. Additionally, they exposed hyperparameters for the online execution (which were hard-coded in previous winner solution) that control, for example, when to evaluate during the submission and the number of samples used. To further increase the potential of the existing solution, they extended the configuration space to also include:

— An EfficientNet Tan and Le (2019) (in addition to *kakaobrain*'sBrain (2019) ResNet-18) pre-trained on ImageNet Russakovsky et al. (2015);
— The proportion of weights frozen when fine-tuning;
— Additional stochastic optimizers (Adam Kingma and Ba (2014), AdamW Loshchilov and Hutter (2019), Nesterov accelerated gradient Nesterov (1983)) and learning rate schedules (plateau, cosine Loshchilov and Hutter (2017));
— A simple classifier (either a SVM, random forest or logistic regression) that can be trained and used within the first 90 seconds of the submission.

After the extension of the configuration space, they optimized the hyperparameters with BOHB Falkner et al. across 300 evaluation runs with a time budget of 300 seconds on eight different datasets (Chucky Krizhevsky et al. (2009), Hammer ham (2018), Munster LeCun et al. (2010), caltech_birds2010 Welinder et al. (2010), cifar100 Krizhevsky et al. (2009), cifar10 Krizhevsky et al. (2009), colorectal_histology Kather et al. (2016) and eurosat Helber et al. (2017)). These eight datasets were chosen from meta-training data to lead to a portfolio of complementary configurations Feurer et al. (2018); Xu et al.. Additionally, they added a robust configuration to the portfolio of configurations that performed best on average across the eight datasets. Then, they evaluated each configuration of the portfolio for 600 seconds on all 21 image datasets they had collected. In addition, they searched for a tenth configuration (again with BOHB), called the *generalist*, that they optimized for the average improvement across all datasets relative to the already observed ALC scores. In the end, the meta-train-data consisted of the ALC performance matrix (portfolio configurations $\times$ datasets) and the meta-features from the 21 datasets. These meta-features consisted of the image resolution, number of classes, number of training and test samples and the sequence length (number of video frames, i.e. 1 for image datasets). In addition, they studied the importance of the meta features for the meta-learner, and selected an appropriate subset. To optimize the portfolio further, they applied a greedy submodular optimization Feurer et al. (2018); Xu et al. (2011) to

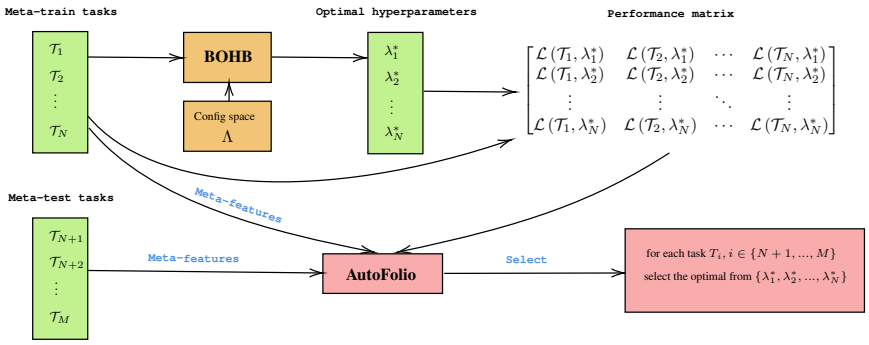


Figure 4.7 **Workflow of *automl_freiburg***. The approach first optimizes the hyperparameter configuration (including choices for training, input pipeline, and architecture) for every task (dataset) in our meta-training set using BOHB Falkner et al.. Afterwards, for each dataset i, the best found configuration λ_i^* is evaluated on the other datasets $j \in \{1,2,...,N\}$, $j \neq i$ to build the performance matrix (configurations $\times$ datasets). For training and configuring the meta-selection model based on performance matrix and the meta-features of the corresponding tasks, the approach uses AutoFolio Lindauer et al. (2015). At meta-test time, the model fitted by AutoFolio uses the meta-features of the test tasks in order to select a well-performing configuration.

minimize the chance of wrong predictions in the online phase. Based on this data, they trained a cost-sensitive meta-model Xu et al. (2012) with AutoFolio Lindauer et al. (2015), which applies algorithm configuration based on SMAC Hutter et al., 2011b) to efficiently optimize the hyperparameters of the meta-learner. Since the meta-learning dataset was rather small, HPO for the meta-learner could be done within a few seconds. Lastly, they deployed the learned AutoFolio model and the identified configurations into the initialization function of the winner's solution code. The workflow of this approach is shown in Figure 4.7.

4.7 . Post-challenge analyses

4.7.1 . Ablation study

To analyze the contribution of different components in each winning team's solution, we asked 3 teams (*DeepWisdom*, *DeepBlueAI* and *automl_freiburg*) to carry out an ablation study, by removing or disabling certain component (e.g. meta-learning, data augmentation) of their

approach. We will introduce in the following more details on these ablation studies by team and synthesize thereafter.

DeepWisdom

According to the team *DeepWisdom*, three of the most important components leading to the success of their approach are: meta-learning, data loading and data augmentation. For the ablation study, these components are removed or disabled in the following manner:

- **Meta-learning (ML)**: Here meta-learning includes transfer learning, pretrain models, and hyperparameter setting and selection. Meta learning is crucial to both the final accuracy performance and the speed of train-predict lifecycle. For comparison we train models from scratch instead of loading pre-trained models for image, video and speech data, and use the default hyperparameter settings for text and tabular subtasks.
- **Data Loading (DL)**: Data loading is a key factor in speeding up training procedures to achieve higher ALC score. We improve data loading in several aspects. Firstly, we can accelerate decoding the raw data formatted in a uniform tensor manner to numpy formats in a progressive way, and batching the dataset for text and tabular data could make the conversion faster. Secondly, the cache mechanism is utilized in different levels of data and feature management, and thirdly, video frames are extracted in a progressive manner.
- **Data Augmentation (DA)**: Fast auto augmentation, time augmentation and a stagewise spec_len configuration for thinresnet34 model are considered as data augmentation techniques for image, video and speech data respectively.

We carried out experiments on the 10 final phase datasets with above components removed. The obtain ed ALC scores are presented in Figure 4.8. As it can be seen in Figure 4.8, Meta-Learning can be considered one of the most important single component in DeepWisdom's solution. Pre-trained models contribute significantly to both accelerating model training and obtaining higher AUC scores for image, video and speech data, and text and tabular subtasks benefit from hyperpa-

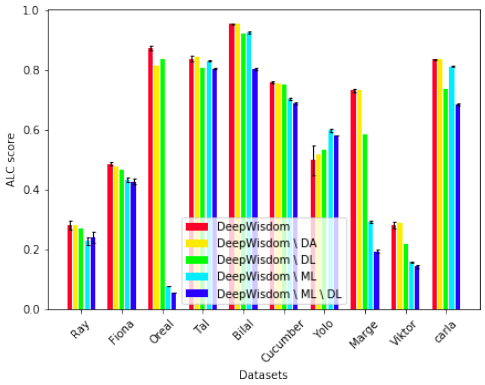


Figure 4.8 **Ablation study for *DeepWisdom*:** We compare different versions of *DeepWisdom*'s approach, with one component of their workflow disabled. "DeepWisdom \ ML" represents *DeepWisdom*'s original approach but with Meta-Learning disabled. "DA" code for Data Augmentation and "DL" for Data Loading. The method variants are ordered by their average rank from left to right. Thus we observe that removing Data Augmentation does not make a lot of difference, while removing both Meta-Learning and Data Loading impacts the solution a lot. See Section 4.7.1 for details.

rameter setting such as model settings and learning rate strategies. For image, we remove pre-trained models for both ResNet-18 and ResNet-9, which are trained on the ImageNet dataset with 70% and 65% top1 test accuracy; for video, we remove the parts of freezing and refreezing the first two layers. Then the number of the frames for ensemble models and replace MC3 model with ResNet-18 model. For speech, we do not load the pre-trained model which is pre-trained on VoxCeleb2 dataset, that is we train the thin-resnet34 model from scratch. For text, we use default setting, i.e. do not perform meta strategy for model selections and do not perform learning rate decay strategy selections. For tabular, with the experience of datasets inside and outside this competition, we found two sets of params of lightgbm. The first hyperparameters focus on the speed of lightgbm trainning, it use smaller boost round and max depth, bigger learning rates and so on. While the second hyperparameters focus on the effect of lightgbm trainning, it can give us a generally better score. We use the default hyperparameters in lightgbm in the minus version.

Data Loading is a salient component for the ALC metric in anytime learning. For text, speech and tabular data, data loading speeds up numpy data conversion to make the first several predictions as quickly as possible, achieving higher alc scores. In the minus version, we convert all train tfdatasets to numpy array in the first round, and alc scores of nearly all datasets on all modalities decrease steadily compared with full version solution.

The data augmentation component also helps the alc scores of several datasets. In the minus version for speech data we use the fixed spec_len config, the default value is 200. Comparison on Marge and Oreal datasets is obvious, indicating that longer speech signal sequences could offer more useful information. Fast auto augmentation and test time augmentation enhance performance on image and video data marginally.

DeepBlueAI

According to the team *DeepBlueAI*, three of the most important components leading to the success of their approach are: adaptive strategies, ensemble learning and scoring time reduction. For the ablation study, these components are removed or disabled in the following manner:

- **Adaptive Strategies (AS)**: In this part, all adaptive parameter settings have been cancelled, such as the parameters settings according to the characteristics of data sets and the dynamic adjustments made during the training process. All relevant parameters are changed to default fixed values.
- **Ensemble Learning (EL)**: In this part, all the parts of ensemble learning are removed. Instead of fusing the results of multiple models, the model that performs best in the validation set is directly selected for testing.
- **Scoring Time Reduction (STR)**: In this part, all scoring time reduction settings were modified to default settings. Related parameters and data loading methods are same as those of baseline.

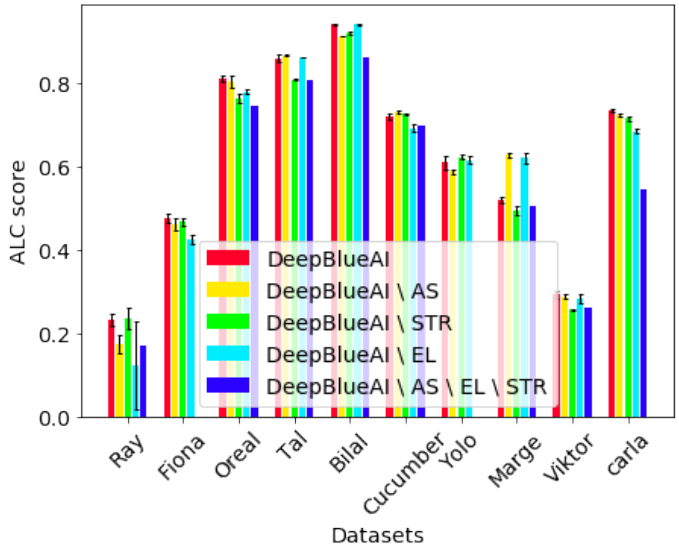


Figure 4.9 **Ablation study for *DeepBlueAI*:** Comparison of different versions of *DeepBlueAI*'s approach after removing some of the method's components. "DeepBlueAI \ AS" represents their approach with Adaptive Strategy disabled. "EL" codes for Ensemble Learning and "STR" for Scoring Time Reduction. For each dataset, the methods are ordered by their average rank from left to right. While disabling each component separately yields moderate deterioration, disabling all of them yields a significant degradation in performance. See Section 4.7.1.

As it can be observed in Figure 4.9, the results of DeepBlueAI have been greatly improved compared with those of DeepBlueAI \AS \EL \STR (i.e., blue bar), indicating the effectiveness of the whole method. After removing the AS, the score of most data sets has decreased, indicating that adaptive strategies are better than fixed parameters or models, and has good generalization performance on different data sets. When STR is removed, the score of most data sets is reduced. Because the efficient data processing used can effectively reduce the scoring time, thereby improving the ALC score, which shows the effectiveness of the scoring time reduction. After EL is removed, the score of the vast majority of data sets has decreased, indicating the effectiveness of ensemble learning to improve the results.

automl_freiburg

According to the team *automl_freiburg*, two of the most important components leading to the success of their approach are: meta-learning and hyperparameter optimization. For the ablation study, these components are removed or disabled in the following manner:

- **Meta-Learning with Random selector (MLR)**: This method randomly selects one configuration out of the set of most complementary configurations (Hammer, caltech_birds2010, cifar10, eurosat).
- **Meta-Learning Generalist (MLG)**: This method does not use AutoFolio and always selects the generalist configuration that was optimized for the average improvement across all datasets.
- **Hyperparameter Optimization (HPO)**: Instead of optimizing the hyperparameters of the meta-selection model with AutoFolio, this method simply uses the default AutoFolio hyperparameters.

As previously mentioned, *automl_freiburg* focused on the computer vision domain (i.e., datasets *Ray*, *Fiona*, *Cucumber*, and *Yolo*). The results of their ablation study, shown in Figure 4.10, indicate that the hyperparameter search for the meta-model overfitted on the eight meta-train-datasets used (original vs HPO); eight datasets is generally regarded as insufficient in the realm of algorithm selection, but the team was limited by compute resources. However, the performance

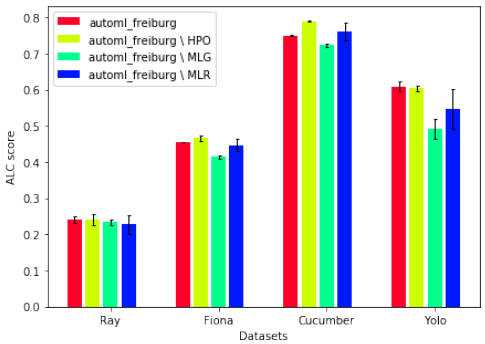


Figure 4.10 **Ablation study for *automl_freiburg*:** Comparison of different versions of *automl_freiburg*'s approach. Since the approach addresses only computer vision tasks, only results on image datasets (*Ray*, *Cucumber*) and video datasets (*Fiona*, *Yolo*) are shown. Average and error bars of ALC scores are computed over 9 runs. "automl_freiburg \ HPO" represents *automl_freiburg*'s approach with default AutoFolio hyperparameters. Likewise, "MLG" stands for the generalist configuration and "MLR" for randomly selecting a configuration from the pool of the most complementary configurations. See Section 4.7.1.

of the non-overfitted meta-model (HPO) clearly confirms the superiority of the approach over the random (MLR) and the generalist (MLG) baselines on all relevant datasets. More importantly, not only does this observation uncover further potential of *automl_freiburg*'s approach, it is also on par with the top two teams of the competition on these vision datasets: average rank 1.75 (*automl_freiburg*) versus 1.75 and 2.5 (*DeepWisdom*, *DeepBlueAI*). The authors emphasize that training the meta-learner on more than eight meta-train datasets could potentially lead to large improvements in generalization performance. Despite the promising performance and outlook, results and conclusions should be interpreted conservatively due to the small number of meta-test datasets relevant to *automl_freiburg*'s approach.

4.7.2 . AutoML generalization ability of winning methods

One crucial question for all AutoML methods is whether the method can have good performances on unseen datasets. If yes, we will say the method has *AutoML generalization ability*. To quantitatively measure this ability, we propose to compare the average rank of all top-8 methods in both feedback phase and final phase, then compute the Pearson cor-

Figure 4.11 **Task over-modeling:** We compare performance in the feed-back and final phase, in an effort to detect possible habituation to the feed-back datasets due to multiple submissions. The average rank of the top-8 teams is shown. The figure suggests no strong over-modeling (over-fitting at the meta-learning level): A team having a significantly better rank in the feed-back phase than in the final phase would be over-modeling (far above the diagonal). The Pearson correlation is $\rho_{X,Y} = 0.91$ and p-value $p = 5.8 \times 10^{-4}$.

relation (Pearson's ρ) of the 2 rank vectors (thus similar to Spearman's rank correlation noa (2020a)). Concretely, let r_X be the average rank vector of top teams in feedback phase and r_Y be that in final phase, then the Pearson correlation is computed by $\rho_{X,Y} = \text{cov}(r_X, r_Y)/\sigma_{r_X}\sigma_{r_Y}$.

The average ranks of top methods are shown in Figure 4.11, with a Pearson correlation $\rho_{X,Y} = 0.91$ and p-value $p = 5.8 \times 10^{-4}$. This means that the correlation is statistically significant and no leaderboard over-fitting is observed. Thus the winning solutions can indeed generalize to *unseen* datasets. Considering the diversity of final phase datasets and the arguably out-of-distribution final-test meta-features shown in Table 3.2, this is a feat from the AutoML community. Thus it's highly plausible that we are moving one step closer to a universal AutoML solution.

4.7.3 . Impact of t_0 in the ALC metric

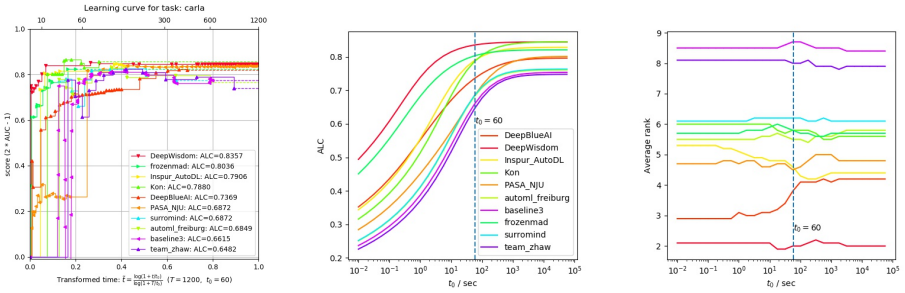
We recall that the Area under Learning Curve (ALC) is defined by

$$\begin{aligned} ALC &= \int_0^1 s(t) d\tilde{t}(t) \\ &= \int_0^T s(t)\tilde{t}'(t)dt \\ &= \frac{1}{\log(1+T/t_0)} \int_0^T \frac{s(t)}{t+t_0} dt \end{aligned} \tag{4.1}$$

where

$$\tilde{t}(t) = \frac{\log(1+t/t_0)}{\log(1+T/t_0)} \tag{4.2}$$

Thus t_0 parameterizes a weight distribution on the learning curve for computing the ALC. When t_0 is small, the importance weight at the



(a) Learning curves for the task *Carla*

(b) Impact of t_0 on the ALC scores for task *Carla*.

(c) Average rank among AutoDL final phase participants, using different t_0. The legend is hidden and is the same as that of Figure 4.12b.

Figure 4.12 **Any-time learning *vs.* fixed-time learning:** We evaluate the impact of parameter t_0 on the ALC scores and the final rank. This parameter allows us to smoothly adjust the importance of the beginning of the learning curve (and therefore the pressure imposed towards achieving any-time learning). When t0 is small, the ALC puts more emphasis on performances at the beginning of the learning curve and thus favors fast algorithms. When t_0 is large, similar weight is applied on the whole learning curve, performances are uniformly averaged, so being a little bit slow at the beginning is not that bad, and it is more important to have good final performance when the time budget is exhausted (fixed-time learning). The tabular dataset *Carla* is taken as example. The fact that two learning curves cross each other is a necessary condition for the impact of t_0 on their ranking on this task. Learning curves of top teams on this dataset are shown in 4.12a. The impact of t_0 on the ALC scores of these curves is shown in 4.12b. We see that when t_0 changes, the ranking among participants can indeed change, typically the ALC of *frozenmad* is larger than that of *Kon* but this is not true for large t_0. In 4.12c, the fact that the average rank (over all 10 final phase datasets) varies with t_0 also implies that t_0 can indeed affect the ranking of ALC on individual tasks. However, we see that the final ranking (i.e. that of average rank) is quite robust against changes of t_0. Very few exceptions exist such as *PASA_NJU* and *Inspur_AutoDL*. Overall, t_0 proved to have little impact, particularly on the ranking of the winners, which is another evidence that top ranking participants addressed well the any-time learning problem.

beginning of the curve is large. Actually when t_0 varies from 0 to infinity, we have

$$\lim_{t_0 \to 0^+} ALC(t_0) = s(0)$$

and

$$\lim_{t_0 \to +\infty} ALC(t_0) = \frac{1}{T} \int_0^T s(t)dt.$$

So different t_0 might lead to different ALC ranking even if the learning curve $s(t)$ is fixed. It is then to be answered whether the choice of $t_0 = 60$ in AutoDL challenge is reasonable. For this, we reflect the impact of t_0 on the ALC scores and the final average ranking in Figure 4.12. Observation and discussion can be found in the caption. We conclude that t_0 does affect the ranking of ALC scores but the final ranking is robust to changes of t_0, justifying the choice of t_0 and the challenge setting.

4.8 . Conclusion

In conclusion, we are encouraged to continue our challenge series in machine learning with code submission and blind testing in a well-defined identical computer environment, with a fixed time and memory budget. The latest one, the AutoDL challenge, helped pushing the state of the art in Automated Deep Learning. Our novel challenge design, with emphasis on “any-time learning", permitted to harvest answers to new questions.

Among other things, the challenge revealed that Automated Deep Learning methods are ripe for modalities such as image, video, speech, and text, but no unified solution emerged across modalities, and Deep Learning remained weaker than other methods for tabular data. This raises the question of developing new universal coding, generic workflows, or universal neural architectures. A step in this direction could be to organize a cross-modal Neural Architecture Search (NAS) challenge, to search for universal architectures. Intensive search in architecture space was impractical with the constrained time budget we provided for the AutoDL challenge, but with one order of magnitude more computational resources, it may be feasible.

Deep Learning methods have earned the reputation of being notoriously slow to train and require prohibitive computational resources in domains such as video processing. Not so anymore with "any-time learning methods" allowing users to stop training early and get reasonable performance. The wining teams succeeded in climbing the learning curve fast, without sacrificing the final performance. Transfer learning (fine tuning of pre-trained models), progressive increase in model complexity, fast data loading, and efficient exploration of data space, were key components to achieve these results.

The post-challenge analyses revealed the importance of meta-learning, through ablation studies conducted by winning teams. The teams demonstrated that generalizing to new unseen datasets is possible, and improves by meta-learning, thus they effectively achieved a form of transfer learning. This calls for further research and we envision that a meta-learning challenge should be organized, to conduct a more controlled study. Several settings have been proposed, including: (1) a challenge on model recommendation, similar to the movie recommendation Netflix challenge, in which a sparse matrix with just a few scores of models on a few datasets is initially provided and the goal is to find as quickly as possible the best performing model on a new dataset; (2) a challenge proposing training tasks and test tasks, aiming at training search agents capable of selecting the best performing models to solve the test tasks; (3) an on-line meta-learning challenge (or life-long-learning challenge) in which tasks are made available sequentially to models, who can retain some "memory" of past tasks to perform better in future tasks.

This challenge was limited to tensor data and multi-label problems. Other steps towards enlarging the scope of automated machine learning include generalizing to more complex data structures. This is partially addressed by the on-going AutoGraph challenge. Generalization to other types of tasks was addressed by the AutoSeries challenge. We intend to keep proposing more diverse types of data and tasks to stimulate the community to make progress.

Lastly, challenges are meant to provide fair and reproducible evaluations removing the inventor-evaluator bias. However, other types of

biases can crop up. One such bias stems from the choice of datasets. As organizers, we had to chose datasets with sufficient modeling difficulty to separate well the participants, yet not a too high intrinsic difficulty. By modeling difficulty, we mean the variance in performance between participants. By intrinsic difficulty we mean (1- the best attainable performance). Neither quantity being available to the organizers at the time of selection of the datasets, they must rely on the performances of the baseline methods to evaluate the difficulty of the tasks and thus the choice may be biased. Yet another type of bias is introduced by the baselime methods provided to the participants (such as Baseline 3 in this challenge).

Beyond research results, our challenges have a long lasting impact since we make available a large number of "public" datasets, and the code of winning solutions.

All in all, we conclude this chapter by listing the major lessons learned from AutoDL challenges and providing links to the following chapters:

— **The winning methods are capable of generalizing on new unseen datasets.** This validates the whole point of finding a universal AutoML algorithm and thus the goal of this thesis;

— **NAS approaches are proven not suited for the any-time learning setting of our challenges, at least within the given time budget limit (less than 40 minutes).** It is also worthy to mention that NAS methods could be done prior to the evaluation process with massive computation and the submission only consists of the found architecture. Even though this process is not reflected in our challenges, the importance of NAS methods is still worth further investigation. We will present an NAS algorithm that falls into this category in Chapter 5.

— **Meta-learning is proven effective to achieve universal AutoML.** However, this observation only comes after post-challenge analysis with ablation study. In Chapter 6, we will introduce a meta-learning challenge in preparation to fairly evaluate and compare the effectiveness of meta-learning for all approaches.

5 - Neural Architecture Search

In this chapter, we introduce *GramNAS*, a framework based on formal grammars that we propose in this thesis as a versatile tool for encoding the *search space* for neural architecture search (NAS). Using the terminology from Chapter 3, most NAS solutions can be considered as a form of first-order meta-learning solution where the meta-dataset has the form

$$\mathfrak{D}_{tr} = \{(T_0, \Psi(\beta_j), R_j) | j \in J\}.$$

One can see that here Ψ plays an important role for the representation of each algorithm β_j and thus will be the major subject of research in this chapter. In addition to proposing a custom algorithm representation method that defines the search space, various heuristic search strategies are also adapted and deployed to explore this search space: Mote-Carlo Tree Search (MCTS) and Evolutionary Algorithms.

5.1 . Motivation and Background

As started in Elsken et al. (2019), the basic ingredients of NAS are:

— search space,

— search strategy, and

— evaluation method.

This chapter tackles the problem of facilitating the formal description of the search space, not precluding from any particular search strategy or evaluation method. To illustrate this point, we show results for two rather different search strategies: MCTS and EA.

The need for a new formal description attracted our attention because of our interest in benchmarks. Although the research on Neural Architecture Search (NAS) (Baker et al., 2017; Cai et al., 2018; Liu et al., 2018a, 2019a; Pham et al., 2018; Zoph and Le, 2016) has been very active in recent years, it remains difficult to reproduce results and fairly compare their performances (Yang et al., 2020). As pointed out in many benchmark works (Ying et al., 2019; Zela et al., 2020), one important

hurdle preventing reproducibility and fair comparisons is the lack of consistency in the formalization of *search spaces*.

The NAS problem is often brought back to a combinatorial optimization problem and as such is amenable to a variety of search strategy, including Reinforcement Learning (Baker et al., 2016; Cai et al., 2017; Negrinho and Gordon, 2017a; Zoph and Le, 2016), Bayesian Optimization (Jin et al., 2019), evolutionary algorithms (Miikkulainen, 2002; Real et al., 2017; Xie and Yuille, 2017), surrogate model-based optimization (Liu et al., 2018a; Luo et al., 2018) according to Wistuba et al. (2019). Differential methods with weight sharing (Chen et al., 2021; Liu et al., 2019a) (sometimes called *one-shot architecture search* (Wistuba et al., 2019; Zela et al., 2020)).

Our approach aims at simultaneously addressing the issues of (1) extensibility and diversity; and (2) uniformity and consistent formalism. The corresponding means are:

1. Extensibility and diversity: Avoiding to rely on a *meta-architecture*, which limits design of search spaces to a finite pre-defined number of possibilities, and constrains the emergence of novel architectures;
2. Extensibility and diversity: Not restricting to "vocabulary" of search (modules and connectors) to a pre-defined finite set.
3. Uniformity and consistent formalism: Avoiding ad-hoc definitions of production rules, without refering to an established theory;
4. Uniformity and consistent formalism: Fostering reproducibility by providing concrete guidelines for writing the search space.

Thus, what we want is a unified framework that allows fair comparison and at the same time, leaves open the possibility for pushing forward the state-of-the-art with novel architectures and more. This is the major motivation behind our proposed method which encodes NAS search spaces with *formal grammars*.

As we know, the theory behind formal grammars lay the ground for modern programming languages and computer science in general. It is thus not surprising that diverse search space encoding paradigms are within the reach of formal grammars. Indeed, we will show in this

chapter that our approach, GramNAS, encompasses the search spaces considered by almost all existing works (e.g. (Ying et al., 2019; Zoph and Le, 2016; Zoph et al., 2018)) and demonstrates satisfying expressivity. With GramNAS, it is thus possible to include previous approaches in the same framework and allow fair comparisons. Furthermore, due to the robustness and expressivity of our framework, we suggest in discussion the possibility to carry out meta-learning that can allow faster search based on prior tasks. Finally, we point out the fact that the tree structures, which naturally arise in the formal grammar context, allow in-depth analysis using metric and distance on graph. In fact, any a tree linking the searched neural network architectures naturally defines a distance/metric on these architectures. This distance can be further used for representing the similarity between different architectures. Jin et al. (2019) used a similar notion (the *edit distance*) and applied it with Bayesian Optimization. Another similar idea on graph distance can be found in Das et al. (2014).

In the following, we will first introduce some basic notions on formal grammars, especially on *context-free grammars*. Then we will show how one can encode search spaces of NAS using the formal grammar framework (GramNAS) and illustrate its expressivity. Lastly, we combine GramNAS with two search strategies (MCTS and evolutionary algorithms) and compare their performances with other related works.

5.2 . Proposed Method

5.2.1 . Encoding Search Space with Formal Grammar

A string along with a given formal grammar (Sipser, 1997) parsing it has always been the natural way to define a neural network architecture.

Whether it be code written in C++, in Python or even mathematical formulas, they all come along with strings structured according to a certain grammar. In addition, almost all modern programming languages (e.g. C++ [1], Python [2]) use a context-free grammar parser (after a

1. https://www.nongnu.org/hcb/
2. https://docs.python.org/3/reference/grammar.html

tokenization step). Thus we propose to use context-free grammars to encode the search space of NAS.

We first introduce some basic notations and definitions, following notations in Sipser (1997). For a finite set $\mathscr{A}$ (the **alphabet**) of characters (or **letters**), we denote $\mathscr{A}^* = \{x_1 x_2 \dots x_k | k \in \mathbb{N} \text{ and } x_i \in \mathscr{A} \forall 1 \leq i \leq k\}$ the set of all strings (also called **words**) concatenatin characters from $\mathscr{A}$. For $k = 0$, the empty string is denoted by ε.

Definition 1. *A **context-free grammar** is a 4-tuple (V, Σ, R, S), where V is a finite set called the **non-terminals** (or variables); Σ is a finite set, disjoint from V, called the **terminals**; R is a finite set of (production) **rules**, with each rule stating that a given variable can be rewritten as a string of non-terminals and terminals; and $S \in V$ is the start variable.*

If $A \in V$, $u, v, w \in (V \cup \Sigma)^*$ and $r = A \to w$ is a rule of the grammar, we say that uAv yields uwv following the rule r, written $uAv \Rightarrow uwv$ (or $uAv \overset{r}{\Rightarrow} uwv$). It is said that u **derives** v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if there exists a sequence $u_1, r_1, u_2, \dots, u_k, r_k$ with $k > 0$ and

$$u \overset{r_0}{\Rightarrow} u_1 \overset{r_1}{\Rightarrow} u_2 \overset{r_2}{\Rightarrow} \dots \overset{r_{k-1}}{\Rightarrow} u_k \overset{r_k}{\Rightarrow} v. \tag{5.1}$$

Such a sequence is called a **derivation** from u to v.

By definition, word v is **terminal** iff all its letters are terminal ($vin\Sigma^*$); one can no more apply any rule on v. For a grammar $G = (V, \Sigma, R, S)$, we denote by $L(G)$ the set of all terminal words of G. We say $L(G)$ is the **language** recognized by G.

In this work, we only consider **leftmost derivations**, for which all rules r_i apply on the *leftmost* non-terminal of the string u_i. Since we will only consider derivations that end by a terminal word, this constraint does not induce any loss of generality. This is because all derivations can be transformed into a leftmost derivation by making a permutation on the order of applying r_i's[3]. In practice, one often uses *tokens* (i.e. strings) to represent non-terminals. For example, the start symbol S can be represented by a string `'start'`. The vocabulary is often omitted since one can always think of a general encoding method such

3. This fact is only valid for context-free grammars, not for context-sensitive grammars.

as Unicode (noa, 2020b). An example of a simple formal grammar is as follows.

```
start: layers

layers: layer layers
      | layer

layer: "<conv2d>"
```

Here we use tokens such as *start*, *layer* and *layers* instead of letters. The above grammar defines a class of feed-forward neural networks with only convolutional layers. Note that this grammar is written in Backus-Naur form (BNF) (wik, 2020a), with non-terminals represented by strings and '|' to write rules with the same starting non-terminal in a more compact way. In this work, all grammars will be written in either BNF or in the Extended Backus-Naur form (EBNF) (wik, 2020b), which are both supported by the Lark parser generator (https://github.com/lark-parser/lark).

Code examples of grammars we used can be found at

https://github.com/zhengying-liu/GramNAS

5.2.2 . Semantics of formal grammars

Formal grammars allow us to encode objects using strings. However, the *meaning* of each string requires further efforts to specify. The process of associating *meaning*, also known as *semantics*, to terms of formal grammars is called *interpretation*. For example, when analyzing the grammar from previous section, one may wonder what *"<conv2d>" mean* exactly. In this case, one may *interpret "<conv2d>"* as a convolutional layer with certain filter size (which could be concretely implemented with say TensorFlow (Abadi et al., 2016) or PyTorch (Paszke et al.)). In our work, we reuse the implementation of the Lark parser framework [4] and the part of work on semantics is done in a separate Python file which defines a Transformer class specifying the transformations during the parsing process.

4. Lark parser: https://github.com/lark-parser/lark

5.2.3 . Expressivity of context-free grammars

Chomsky hierarchy

Chomsky hierarchy (Chomsky, 1956) categorizes all formal grammars into four categories, namely Type 0, Type 1, Type 2 and Type 3. Corresponding languages are called *recursively enumerable languages*, *context-sensitive languages*, *context-free languages* and *regular languages*. Each language type is (strictly) included in previous types, as shown in Figure 5.1.

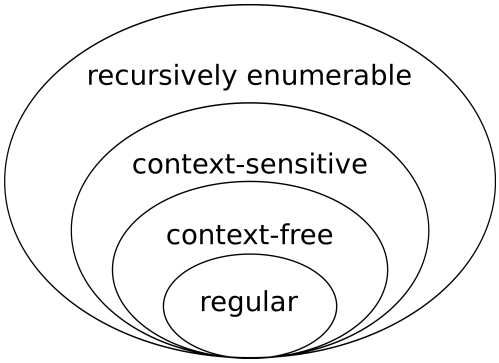


Figure 5.1 **Chomsky hierarchy for formal grammars/languages.**

Regular languages are those that can be recognized by regular expressions. This includes all integers, floating numbers, URLs, etc. In the following, we will introduce corresponding grammars that encode these sets to enable further usage in our proposed NAS approach.

Encoding natural numbers and floating numbers

One can encode the set of all natural numbers [5] by following context-free grammar in Figure 5.2.

We note that this grammar is actually *regular* since one can write all rules in one of the following three forms: $A \to \varepsilon$, $A \to a$ or $A \to Ba$, where $A, B \in V$ and $a \in \Sigma$.

Similarly, one can have a context-free grammar for decimal numbers, as shown in Figure 5.3.

5. Here we allow beginning zeros for illustrative purpose. One can easily avoid this by using some more specific rules.

```
start: integer

integer: integer digit
       | digit

digit: "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Figure 5.2 **Grammar for encoding natural numbers**

```
start: decimal

decimal: sign integer "." integer

sign: "" | "-"

integer: integer digit
       | digit

digit: "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

Figure 5.3 **Grammar for encoding decimal numbers**

Note that the non-terminal *integer* can actually be reused from above. And in the future, we will omit the rules for terms such *integer* when the context is clear.

Encoding directed graphs

Directed graphs are very important objects in computer science and in particular, very relevant to our work since neural network architectures are often under a form of directed graph. We show how to encode the set of all directed graphs (each with a finite number of nodes) using a context free grammar.

Given a directed graph $G = (V, E)$, we can order its nodes and use an index (an integer) to represent each node. Thus for a graph with n nodes, we can always assume without loss of generality that $V = \{0, 1, ..., n-1\}$. Then a directed graph can be completely characterized by the lists of out-going edges for each node. Each list is then a list of integers. Hence

the following grammar gives a possible encoding of directed graphs. Here we omit the rules for *integer* (reused from the definition above).

```
start: graph

graph: "[" lists "]"

lists: list "," lists
     | list

list: "[" integers "]"

integers: integer "," integers
        | integer
        | ""

integer: [...]
```

Figure 5.4 **Grammar for encoding directed graphs**

As an example, a valid word of this grammar is

`[[1,3],[3],[3],[]]`

which specifies a directed graph with 4 nodes, as shown in Figure 5.5. Note that this grammar allows graphs with *repetitive* edges (there is

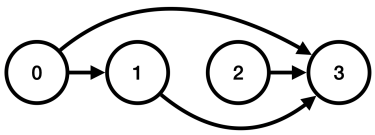


Figure 5.5 **Directed graph specified by the given string `[[1,3],[3],[3],[]]` and the given grammar.**

nothing preventing a same integer to appear twice in a list). This issue can be avoided using another grammar, such as the one shown in Figure 5.6.

which encodes the list of arriving nodes by a string of 0's and 1's as positional code, instead of a list of integers. Then one possible string that encodes the graph in Figure 5.5 will be

```
start: graph

graph: "[" lists "]"

lists: str "," strs
     | str

str: (0|1)*
```

Figure 5.6 **Grammar for encoding directed graphs, without repetitive edges.**

```
[0101,0001,0001,0000]
```

In the following, we will still focus on the first grammar with list of integers for simplicity, as the issue of repetitive edges does not harm the expressivity of the language.

Encoding directed acyclic graphs (DAG)

For classification or regression tasks, most neural network architectures are feed-forward networks with no recurrent units or any loop. In this case, NN architectures, when considered as directed graphs with neurons (or layers) being nodes and connections edges, are *directed acyclic graphs* (DAG). As an example, we use the formulation of (Elsken et al., 2019) and consider layers as nodes in the following. Let $L_0, L_1, ..., L_n$ be the layers of an NN architecture, with L_0 being the input layer and L_n being the output layer. Each layer takes the output from other layers and perform some operations such as addition, multiplication or concatenation. For each layer L_i, let L_i^{out} denote its output after its operation. Then to guarantee that there is no cycle among the layers, we only allow L_i^{out} to be a function of *previous* layers, i.e.

$$L_i^{out} = g_i(L_{i-1}^{out}, L_{i-2}^{out}, ..., L_0^{out}), \quad i = 1, ..., n, \tag{5.2}$$

where g_i is some function defining the operation of the layer L_i. (Elsken et al., 2019) dubs this very general representation of neural network architecture *multi-branch networks*. An example architecture is shown

in figure 5.7. We observe that Equation 5.2 is equivalent to saying that

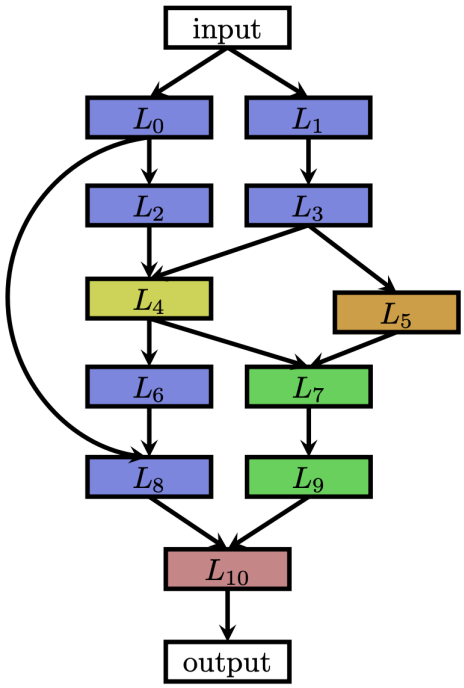


Figure 5.7 Multi-branch networks defined in (Elsken et al., 2019) are equivalent to the fact that the underlying graph of the architecture is a directed acyclic graph (DAG).

the layers $L_0, ..., L_n$ are aligned in a *topological ordering*. Given a graph $G = (V, E)$, a topological ordering is defined to be an ordering of all nodes in which for each edge $(u, v) \in E$, u always comes before v. It is known that the existence of topological ordering is actually equivalent to saying that the graph is a DAG; see e.g., (Kahn, 1962).

Once we represent a graph using a topological ordering, we can use an idea very similar to that in previous section to encode DAGs using formal grammars. Given a graph with n nodes, we again suppose the nodes are the set $V = \{0, ..., n-1\}$. Then this graph is determined by a list of lists of arriving nodes. Concretely, this means a node list $\ell_i \subseteq V$ for each node $i \in V$. Since this is a topological ordering, we have

$$i < j, \forall j \in \ell_i. \tag{5.3}$$

If we make following transformation on ℓ_i and define

$$\tilde{\ell}_i := \{\tilde{j} := j - i - 1 | j \in \ell_i\}, \tag{5.4}$$

then this gives a 1-1 map on all ℓ_i and $\tilde{\ell}_i$ and thus on all DAGs. This means that one can also use the representation from Equation 5.4 to encode any DAG. As the $\tilde{j}$'s are just natural numbers, this allows us to use the *exact same* grammar as that of directed graphs (Figure 5.4), while taking the above transformation into account when interpreting (i.e. defining the semantic).

As an example, we give the string under this grammar (Figure 5.4 but with a different semantic) for the graph in Figure 5.5:

`[[0,2],[1],[0],[]]`

We notice the difference between above string and the string in Figure 5.5 (`[[1,3],[3],[3],[]]`), due to a different semantic defined by the transformation in equation 5.4.

To summarize, we managed to encode the set of all DAGs using a context-free grammar, which is the same as in previous section on directed graphs but with a different interpretation.

5.2.4 . Example search spaces encoded with a grammar

In this section, we show that many popular search spaces in the NAS literature can all be encoded with formal grammars.

Search space from the original NAS paper

As one of the first works in the current NAS literature, Zoph and Le (2016) has a relatively simple search space, shown in Figure 5.8.

Each neural network in the search space consists of several convolutional layers. Each layer is defined by filter height, filter width and the number of filters [6]. Possible choices for filter height and width are 1, 3, 5 and 7, and those for number of filters are 24, 36, 48 and 64. ReLU activation (Nair and Hinton, 2010), batch normalization (Ioffe and

6. The stride height and width are fixed to be 1 in the first experiments of their work.

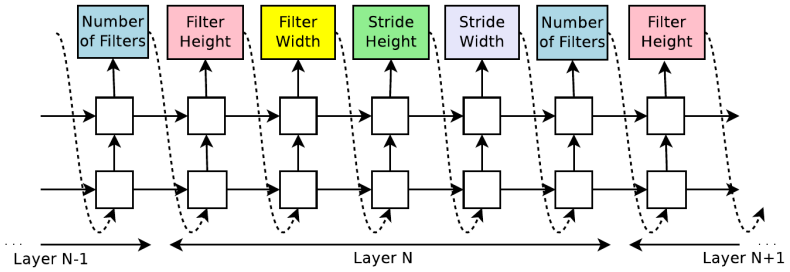


Figure 5.8 Search space defined in Zoph and Le (2016). Each neural network in the search space consists of several convolutional layers. Each layer is defined by filter height, filter width and the number of filters.

Szegedy, 2015) are applied after each layer. Before the output layer, a fully connected layer is applied.

We can encode this simple search space with following grammar, in a straight-forward manner [7].

```
start: layers

layer: "[" "<conv2d>" filter_height filter_width num_filters "]"

filter_height: /1|3|5|7/

filter_width: /1|3|5|7/

num_filters: /24|36|48|64/

layers: layer layers
      | layer
```

Figure 5.9 **Grammar for encoding the search space of the first NAS paper (Zoph and Le, 2016).**

The fact that one can use an arbitrarily large number of layers is implemented by the last two rules for the non-terminal *layers*. Note

7. The implementation can be fount at https://github.com/zhengying-liu/GramNAS/blob/master/gramnas/examples/NAS/nas.lark

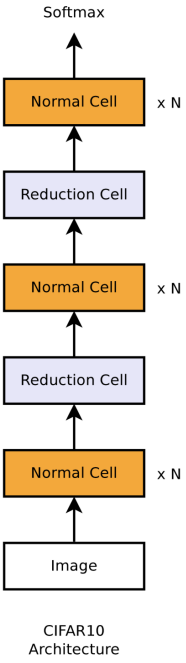


Figure 5.10 **Neural network architectures in NASNet Space (Zoph et al., 2018) for CIFAR-10 dataset.** N is chosen to be 4 or 6.

that we used `/1|3|5|7/` instead of `1|3|5|7` due to the syntax of Lark parser.

NASNet search space

A popular choice of NAS search space is the *NASNet space* (Zoph et al., 2018), which defines two types of *cell*s: Normal Cell and Reduction Cell. Normal Cells do not modify the image dimension (although the number of filters may vary) while Reduction Cells reduce the image dimension (by for example applying max pooling). Then for the search space, one focuses on the structure of these 2 cells. Once these two cell structures are chosen, Normal Cells and Reduction Cells are stacked several times to form a complete architecture, as shown in Figure 5.10.

For determining the structure inside a cell (either a Normal Cell or a Reduction Cell), the procedure goes as follows. Each cell receives as input two hidden states h_{i-1} and h_i from two previous cells. Then in each iteration, two states (not necessarily different) are chosen from existing states. Each of both states is applied with a unary operation

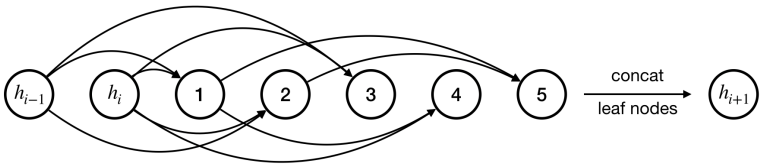


Figure 5.11 Search for cells in NASNet space. The architecture generating procedure is equivalent to iteratively generating nodes of a DAG, while each new node (apart from the two initial ones h_{i-1} and h_i) must have two parent nodes (though they can be the same). Then all nodes with no children (i.e. all leaf nodes) are concatenated along the filter dimension to give the output hidden state h_{i+1}. Here the number of newly generated nodes is chosen to be $B = 5$, as in (Zoph et al., 2018).

(from a predefined list). Then the two output states are combined using element-wise addition or concatenation along the filter dimension. Finally the resulting state is added to existing list of states before beginning next iteration. An example of this procedure is shown in Figure 5.11. We see that this procedure is very similar to that of generating a DAG, except that each newly generated nodes must have two parent nodes.

Now we show that this search space is also a special case of our proposed language. Specifically, one possible formal grammar that encodes this search space is given in Figure 5.12, where the rules for *u_op* have been omitted; *u_op* involves a pre-defined list of unary operations:

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv.

Then *integer* is omitted as usual. However we note that the integers here require more careful treatment, e.g. with a modulo operation to

```
start: nodes

nodes: node "," nodes
     | node

node: "[" parent "," u_op "," parent "," u_op "," bi_op "]"

par: integer

u_op: [...]

bi_op: "add"
     | "concat"

integer: [...]
```

Figure 5.12 **Grammar for encoding the NASNet search space (Zoph et al., 2018).**

guarantee that the parent nodes are from previous nodes. This can be easily implemented in the semantic of the grammar.

The above two examples confirm that grammars can generate diverse structures. We will show more example search spaces in the Section 5.3 of experimental results.

Expressivity of GramNAS

In conclusion, our proposed method GramNAS has satisfying expressivity. As GramNAS can encode directed acyclic graphs (DAG), it can encode chain-structured search space and multi-branch search space with skip connections (Zoph and Le, 2016), which are both special case of DAGs. Additional labels or types of different nodes or edges can be implemented with addition non-terminals with a finite number of terminal values. Cell-based search space can also be encoded with GramNAS with different cell types represented by corresponding non-terminals.

Up till now, we have shown that GramNAS is expressive enough to encode the search space of almost all existing approaches, while bring everything to a unified framework. It opens then the possibility to fairly

compare approaches within this framework and furthermore lays a firm ground for meta-learning or life-long learning.

5.2.5 . Random generation of valid words

Once a grammar encoding the search space is given, one can randomly sample valid words based on the grammar by applying leftmost derivation consecutively until one hits a terminal word. Any sampling method can not only be used to make sanity check of the implementation but also be used as a sub-routine (the roll-out) of MCTS that we will introduce later.

For each leftmost derivation, there is a finite set of actions, namely the set of rules that has the current leftmost non-terminal A as left hand side letter. We will denote this set by R_A. A naïve sampling approach could be: at each leftmost derivation for a non-terminal A, choose the action *uniformly* with respect to R_A. However, as the number of non-terminals can increase explosively, this sampling method can easily be never-ending. For this reason, we use a distribution (a Boltzmann distribution) that favors actions generating less new non-terminals:

$$p(r|r \in R_A) \propto \exp(-\beta \cdot \ell(r)) \tag{5.5}$$

where β is a pre-defined parameter and $\ell(r)$ is the number of non-terminals on the right hand side of the rule, i.e. the number of newly generated non-terminals following the rule r.

In our experiments, we uniformly sample a $\beta \in [0,3]$ at each step of word generation. And we observed termination through out our experiments, for all involved grammars.

5.2.6 . MCTS as Search Strategy

As tree structures arise very naturally with formal grammars, we first propose to apply Monte-Carlo Tree Search (MCTS) as search strategy. MCTS extends the celebrated multi-armed bandit algorithm (Auer, 2003) to tree-structured search spaces. It has already been applied to AutoML in 'classic' machine learning (Rakotoarison et al.), natural language processing (Kumagai et al., 2016) and also in NAS (Wang et al., 2018). But to

our knowledge, our approach is the first to combine formal grammars and MCTS to apply to NAS.

Multi-armed bandit problem

A *multi-armed bandit* (or k-*armed bandit* in this case) is defined to be $B = \{R_1, \ldots, R_k\}$, a set of k real distributions. Each R_i corresponds to the *reward* when pulling the i-th arm. Let $\mu_1, \ldots, \mu_k$ to be the expected values associated with these distributions and let $\mu^* = \max_{i=1,\ldots,k}\{\mu_i\}$. Consider a gambler who iteratively pulls one arm per round and observes the associated reward. The objective is to maximize the sum of the collected rewards (or the *return*). As the gambler has no information on the k distributions, he cannot just pull the arm corresponding to μ^* in every round. After playing T rounds, we define the *regret* to be

$$\rho = T\mu^* - \sum_{t=1}^{T} \hat{r}_t \tag{5.6}$$

where $\hat{r}_t$ is the reward observed in round t. Then equivalently, the objective is to minimize this regret. As a classic solution to the multi-armed bandit problem, the UCB1 algorithm (Auer, 2003) defines such an arm selection policy, selecting in each round the arm i that maximizes the following value:

$$\hat{\mu}_i + \sqrt{\frac{2\log n}{n_i}} \tag{5.7}$$

where $\hat{\mu}_i$ is the sampled average obtained by playing arm i, n_i is the number of times arm i has been pulled and $n = \sum n_i$ is the total number of rounds so far. (Note that, although UCB1 provably yields an optimal logarithmic regret, it is not optimal; the reader is referred to (Bubeck and Cesa-Bianchi, 2012) for a more comprehensive presentation).

We note that the above multi-armed bandit problem is equivalent to a *Markov decision process* (Sutton and Barto, 2018) with one single state. However, the problems we will consider often involve more than one state, which means that the bandit B_s might be different in different state s and when one makes an action (e.g. pulls an arm) there might be a transition of state. When solving this more general Markov decision process problem, one can generalize bandit algorithms (such as the

UCB1 algorithm) to the Monte-Carlo Tree Search (MCTS) algorithm in the following way.

Monte Carlo Tree Search algorithm

The MCTS algorithm maintains a tree of states, with root node corresponding to the initial state, and different states are connected via the action that does the transition. At the beginning, there is only the root node in the tree. Then the MCTS algorithm iterates four phases (Chaslot et al., 2008): selection, expansion, playout (or roll-out / simulation) and back-propagation, illustrated in Figure 5.13:

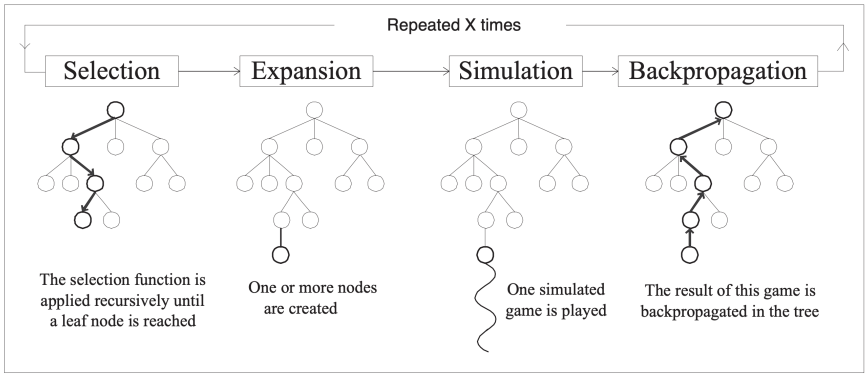


Figure 5.13 **Outline of a Monte-Carlo Tree Search**, figure from (Chaslot et al., 2008).

— **Selection**: In each node of the tree, the child node is selected following a multi-armed bandit strategy, e.g. the UCB1 algorithm or more generally the UCT (Upper Confidence bound applied to Trees) algorithm (Kocsis and Szepesvári, 2006) selects the child node such that it maximizes:

$$\hat{\mu}_i + c\sqrt{\frac{\log n}{n_i}}, \tag{5.8}$$

where, similar to previous section, $\hat{\mu}_i$ is the value of node i, n is the number of times the parent node was visited, n_i is the number of times node i is visited, and c is a problem-dependent constant which balances exploration and exploitation.

— **Expansion**: When a leaf node is reached, an expansion is required. We add the first encountered node that has not been visited to the tree. One can also add more than one nodes by adding first several such nodes.
— **Playout** (or roll-out): When reaching the limits of the visited tree, a roll-out strategy is used to select the options/actions until reaching a terminal node and computing the associated reward. This part explains the name *Monte-Carlo* in that actions are taken randomly until reach a terminal node, simulating a complete episode.
— **Back-propagation**: The reward value is propagated back, i.e. it is used to update the value associated to all nodes along the visited path up to the root node. Typically, all values such as $\hat{\mu}_i$, n and n_i of these nodes need to be updated.

To apply MCTS to a search in a seach space defined by a grammar, we can consider the process of leftmost derivation as a Markov Decision Process (Sutton and Barto, 2018). Then there is only a finite number of actions to make at each step. The state is the current (non-terminal) architecture configuration and the reward is the classification accuracy at the final step (with a complete architecture configuration) or zero otherwise. Then the problem of searching a good architecture becomes an RL problem [8]. We dub this approach combining formal grammars and MCTS **GramNAS**. In the future, we will see that MCTS is far from being the only possibility as search strategy and we will demonstrate this with an evolutionary strategy (AgEBO) in the next section. We will also consider this approach as an example of GramNAS. To distinguish these two methods, we will name them *GramNAS-MCTS* and *GramNAS-AgEBO*.

5.2.7 . Evolution as Search Strategy: the AgEBO algorithm

In addition to MCTS, we also show the possibility of combining other search strategies with formal grammars. Specifically, we adopt an evolutionary algorithm based on Real et al. (2019) and apply it on 4

8. Actually the solution of an RL problem is a *policy* instead of a *state*. But in our case, we can generate the final state as soon as we have learned a good policy. And this final state can serve as the solution to the NAS problem

large tabular datasets from OpenML (Vanschoren et al., 2014) for our study. We note that although our approach is based on recent research works, combining evolutionary algorithms (or genetic programming) and formal grammars have a long history which can date back to the 1990s (Geyer-Schulz, 1995; Ratle and Sebag, 2000; Whigham, 1995; Wong and Leung, 1997) and a relatively recent survey can be found in (Mckay et al., 2010).

Figure 5.14 represents the search space in the form of a graph. It includes the main backbone consisting of repeatable stacks. Each stack consists of two sub-modules: $\mathcal{N}_1$ (either identity or some fully connected dense layers with various number of connections and various activation functions), and similarly constructed sub-module $\mathcal{N}_2$. On either side of the backbone, we have optional skip-connection modules: $\mathcal{SC}_1^2$, $\mathcal{SC}_2^3$, $\mathcal{SC}_1^3$. This is expressed by selecting either 0 or identity for the connection (SC = skip-connections). The nodes shown in red are used to manage the different tensor sizes. For details, please refer to the paper Egele et al. (2021).

With respect to searching, the graph skeleton is fixed. Hence the grammar of this search space is particularly simple: all derivations terminate within 2 steps. Non-terminal nodes include neural network layers ($\mathcal{N}_1$, $\mathcal{N}_2$), skip-connections ($\mathcal{SC}_1^2$, $\mathcal{SC}_2^3$, $\mathcal{SC}_1^3$).

To perform neural architecture search, the main algorithm is aging evolution (Real et al., 2019), as described in black in the pseudo-code of Algorithm 1. The aging evolution algorithm is a particularly simple evolutionary algorithm based on the first-in-first-out (FIFO) principle. Each parent has a single offspring. The successive generations are kept in a queue of fixed length. When the number of individuals (candidate neural network architectures) exceeds the length of the queue, the oldest one is removed. Mutations consist in altering one of the terminal nodes in the grammar, one at a time.

One contribution of the approach is to do jointly neural architecture search (a) and hyperparameter/model search (m) (in blue text in Algorithm 1). The joint neural architecture and hyperparameter search space H is indeed subdivided into H_a and H_m. The problem of joint neural architecture and hyperparameter search is formulated as a bilevel

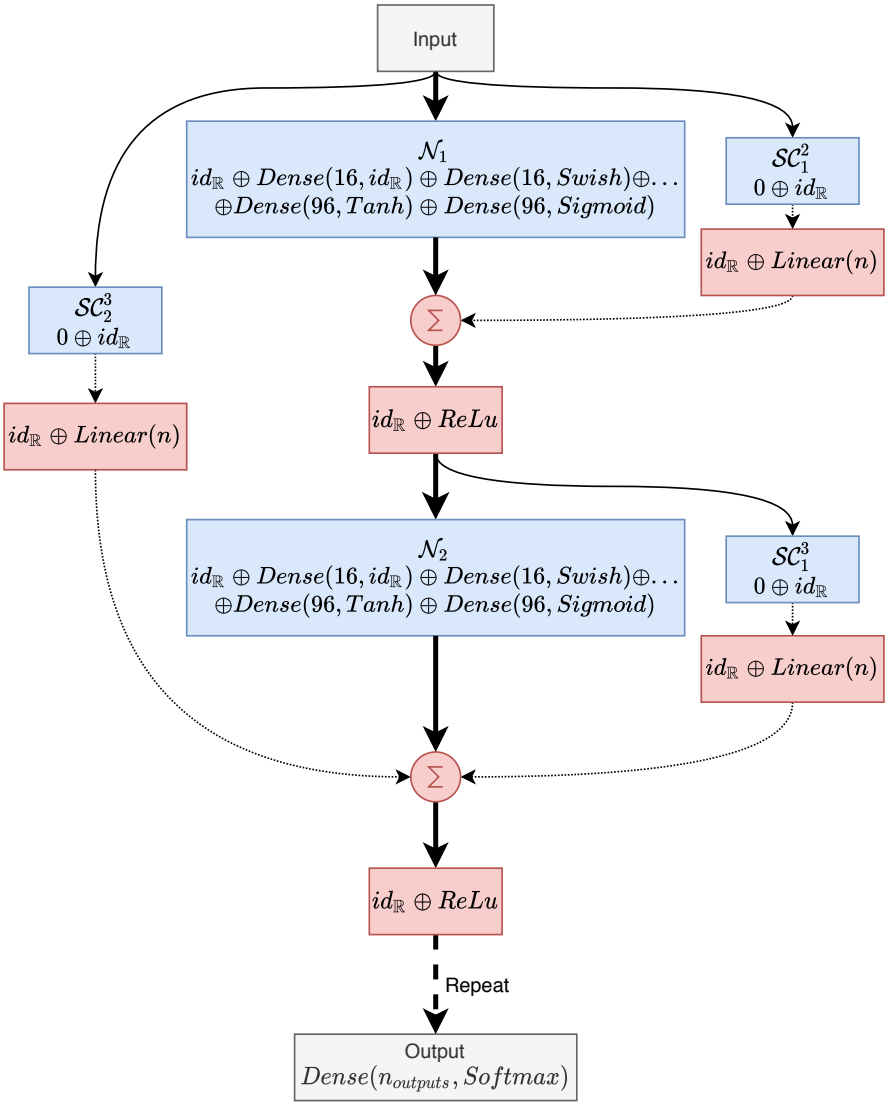


Figure 5.14 **Neural architecture search space for AgEBO**. The nodes $\mathcal{N}_1$ and $\mathcal{N}_2$ represent dense layers $Dense(x, y)$, where x is the number of neurons and y is the activation function. The nodes $\mathcal{SC}_1^2, \mathcal{SC}_1^3, \mathcal{SC}_2^3$ represent the possible skip-connection nodes, when $id_{\mathbb{R}}$ is chosen for each of them. The node $\mathcal{N}_2$ is connected to input node through $\mathcal{SC}_1^2$. The output node is connected to input and $\mathcal{N}_1$ nodes through $\mathcal{SC}_1^3$ and $\mathcal{SC}_2^3$, respectively. The nodes shown in red are used to manage the different tensor sizes and apply an element-wise sum (represented by the plus symbol inside a circle). This stack is repeated 10 times to construct the complete architecture.

optimization problem:

$$
\begin{aligned}
h_a^*, h_m^* &= \underset{(h_a, h_m) \in H_a \times H_m}{\arg\max} \; \mathcal{M}_{w^*}^{val}(h_a, h_m) \\
s.t. w^* &= \underset{w}{\arg\min} \, \mathcal{L}_{h_a, h_m}^{train}(w),
\end{aligned}
\tag{5.9}
$$

```
start: "N:" N ";" N "SC:" SC ";" SC ";" SC

N: "id" | dense

dense: "Dense(" n_neurons "," activation ")"

n_neurons: /16|32|48|64|80|96/

activation: "id_act" |"relu" | "swish" | "tanh" | "sigmoid"

SC: "0" | "id"
```

Figure 5.15 **Grammar for encoding the search space of AgEBO algorithm (Egele et al., 2021).**

where $\mathscr{M}^{val}_{w^*}(h_a, h_m)$ is accuracy on a validation accuracy (measured on a validation subset of the training data) and $\mathscr{L}^{train}_{h_a,h_m}(w)$ is a training accuracy (measured on a training subset of the training data). Test data are used only for final evaluation.

However, at this stage, this two-level procedure is not incorporated in the grammar. For more details, see the paper (Egele et al., 2021).

5.3 . Experimental Results

This experimental section is divided in three parts. In the first two, we report results on a classical benchmark called NAS-Bench-101 (Ying et al., 2019), consisiting of a well-known small-image classification problem (CIFAR-10) Krizhevsky (2009). We first re-use the pre-computed evaluations performed on a pre-defined finite number of combinations of architectures (with fixed hyper-parameters). We then perform a "real" search, including production of new architectures and their evaluation. Finally, in the last subsection, we present results on tabular data, of a search method combining neural architecture search and hyper-parameter selection, using evolution as search strategy.

A toy example on real number approximation is given in Appendix A as a proof-of-concept.

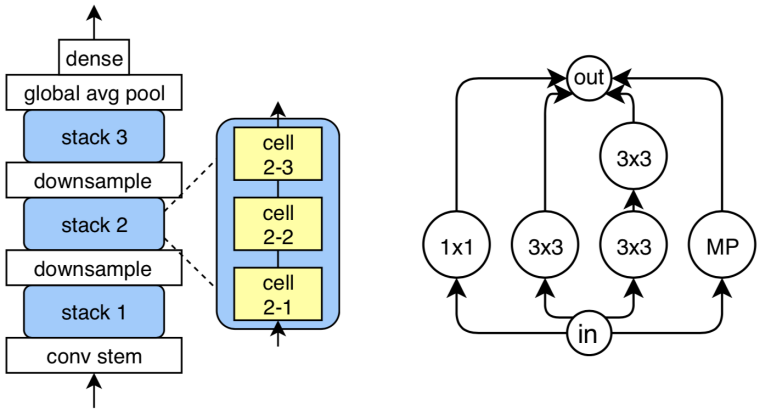


Figure 5.16 **Search space defined by NAS-Bench-101 (Yang et al., 2020).** Left: Each architecture consists of 3 stacks, each of which composed of 3 cells. All cells (and stacks) share the same architectures. Each cell contains a sub-network consisting of five nodes (among three types called "labels" and denoted by '3x3' (3×3 convolution), '1x1' (1×1 convolution) and 'MP' (3×3 max-pooling)) plus the input and output nodes, interconnected according to a 7x7 adjacency matrix, to be determined by search. Right: Example of (the internal structure of) a cell in this search space.

5.3.1 . GramNAS-MCTS applied to NAS-Bench-101 benchmark

We present a first application of GramNAS using NAS-Bench-101 (Ying et al., 2019).

This benchmark consists of pre-computed performances on a variety of convolutional neural networks similar to NASNet (Zoph et al., 2018), shown in Figure 5.12. The search space is based on the structure shown in Figure 5.16. It consists of a fixed backbone of three stacks of three cells. The cell architectures are the only degrees of freedom and the same cell architecture is applied everywhere (different than that of AgEBO). According to the authors, there are approximately 423k (unique) graphs in the search space after de-duplication.

We refer the reader to the NAS-Bench-101 paper (Ying et al., 2019) for more technical details.

Our first task was to formulate the pre-defined search space as a grammar. Then we used MCTS to carry out the search. The grammar is shown in Figure 5.17.

```
start: edges | labels

labels: [label]*5

label: "1x1" | "3x3" | "MP"

edges: [edge]*21

edge: "0" | "1"
```

Figure 5.17 **Grammar for encoding the search space of NAS-Bench-101 (Ying et al., 2019).** We use `[label]*5` to denote `label | label | label | label | label`, i.e. repeating the non-terminal `label` five times, connected by '|'. Idem for `[edge]*21`.

Aside some technicalities (such as a restriction on the number of edges within each cell) that can be found in the code [9], the grammar is relatively simple, as shown in Figure 5.17. It just encodes searching through the architecture of the cells, since the overall structure of the neural network is fixed (Figure 5.16, Left). We note that the adjacency matrix needs only to be filled half-way because the cell architecture is necessarily a DAG, hence only 21 = $\binom{7}{2}$ edges are needed of the 7x7 adjacency matrix.

For each architecture in the search space, NAS-Bench-101 provides the classification accuracy on the CIFAR-10 (Krizhevsky, 2009) dataset after a training of 4, 12, 36 or 108 epochs. The authors provide an API to easily query the accuracy along with the training time by giving an adjacency matrix and a list of 3 labels. So no training is needed but the training time can be recorded to compare the efficiency of different methods.

We evaluated random search (rs), regularized evolution (re) (Real et al., 2019) and our approach GramNAS with MCTS (mcts). Random search consists of randomly generating architectures following the method introduced in 5.2.5, evaluating it and then return the architecture with maximum validation accuracy in the end. Regularized

9. https://github.com/zhengying-liu/GramNAS/blob/master/gramnas/examples/NasBench/env_mcts_vanilla.py

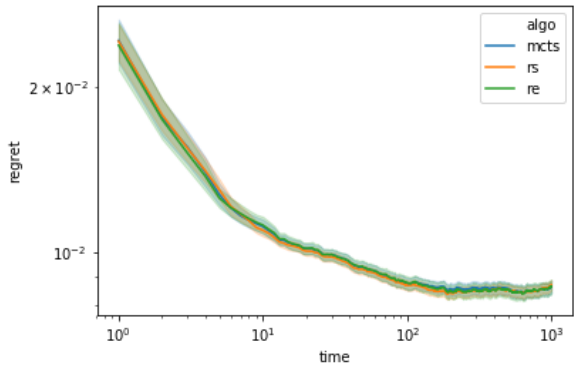


Figure 5.18 **Comparison of NAS methods on NAS-Bench-101 benchmark.** The NAS methods are: our approach (mcts), random search (rs) and regularized evolution (Real et al., 2019) (re). Errors bars are computed over 10 runs. Regret is the difference between the current best obtained test accuracy of the search and the global optimum. Time is in seconds.)

evolution is the same as that introduced in AgEBO (the black part in Algorithm 1).

The results are shown in Figure 5.18. We follow Ying et al. (2019) to demonstrate results using regret vs training time. The *regret* is defined to be the difference between the current best obtained test accuracy of the search and the global optimum (which is known in the NAS-Bench-101 search space, around 94% of accuracy). Time is in second and corresponds to the total training time of all evaluated architectures so far. The results show that no significant difference between the 3 methods is found. On one hand this shows the effectiveness of our approach GramNAS (94% accuracy on CIFAR-10 dataset) but on the other hand, the fact that even random search can obtain similar performances suggests that the problem is too simple, due to the fact that the search space is overly constrained.

5.3.2 . GramNAS-MCTS with a grammar based on NAS-Bench-101

In this section, we show the results of applying GramNAS-MCTS to a search space different to that of NAS-Bench-101 but is inspired by it. As the architectures we consider might not be included in the NAS-Bench-101 search space, we cannot use the performance query API of Ying

et al. (2019). So for each architecture to be evaluated, we train it from scratch with a TensorFlow (Abadi et al., 2016) implementation.

We now present more details on the search space and the corresponding grammar in the following. We consider the best cell found in the NAS-Bench-101 search space, which is shown in Figure 5.19.

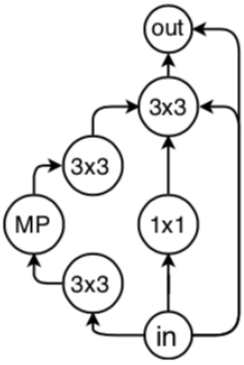


Figure 5.19 **Best cell architecture found in the NAS-Bench-101 search space. Figure from Ying et al. (2019).**

We wish to explore the architectures that use this cell as sub-routine. The grammar is shown in Figure 5.20. Each architecture has the same backbone as in the left figure of Figure 5.16, except that there is no restriction on the number of stacks and the number of cells in each stack. The terminal `nasb_best` corresponds to the best cell found in the NAS-Bench-101 search space (Figure 5.19). The module `reduction` corresponds to the 'downsample' module in Figure 5.16. And the module `nasb_best_dc` is the best cell with doubled number of channels (convolutional filters).

For each architecture in our search space, we train it from scratch on the CIFAR-10 dataset (Krizhevsky, 2009). The number of epochs is chosen to be 1 and the training is done with Adam optimizer Kingma and Ba (2014) of default hyperparameters (e.g. with a learning rate of 0.001). The accuracy on the validation set is used as reward for MCTS.

We compare the performances of our approach with two baselines: DARTS (Liu et al., 2019a) and random search within the same grammar. DARTS is a differentiable NAS method where the (hard) choice of labels

```
start: stacks

stacks: "[[" stacks "-" stack "]]"
      | cells

stack: "reduction" "[" nasb_best_dc "-" cells "]"

// Add "[" and "]" for disambiguity
cells: "[" cells "//" cells "]"
       | "[" cells "-" cells "]"
       | module

cell: "id"
      | "nasb_best"

// Double channels (right after reduction)
nasb_best_dc: "nasb_best_dc"
```

Figure 5.20 **Grammar for encoding a search space based on the best cell architecture found in NAS-Bench-101** (Ying et al., 2019). The terminal `"nasb_best"` corresponds to the best cell shown in Figure 5.19.

(e.g. 1x1, 3x3 or MP) for each layer is softened using a softmax function

$$\bar{o}(x) = \sum_{o \in \mathcal{O}} \frac{e^{\alpha_o}}{\sum_{o' \in \mathcal{O}} e^{\alpha_o}} o(x) \tag{5.10}$$

where $o \in \mathcal{O}$ is an operation such as choosing 3x3 convolution for a layer. Then the parameters α_o are optimized using gradient descent on the validation set. Finally an argmax over α_o is performed to obtain the final architecture. Another baseline we used is random search, which as before generates random architectures according to Section 5.2.5 and return the architecture with best validation accuracy so far.

The results are shown in Figure 5.21. The performance of random search and GramNAS are inferior to that of DARTS, at least before 70000 seconds. However there is no significant difference between GramNAS and random search. This means that the advantages of MCTS might

not be unleashed with very few number of iterations (only up to 80 iterations).

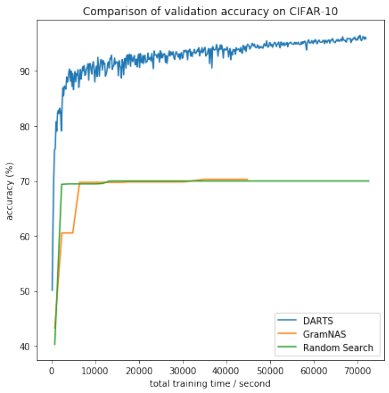


Figure 5.21 **Comparison of GramNAS to DARTS and random search on CIFAR10.** We see that the performance of random search and GramNAS are inferior to that of DARTS, at least before 70000 seconds. There is no significant difference between GramNAS and random search.

5.3.3 . GramNAS-AgEBO on Four Tabular Datasets

Now we apply GramNAS with AgEBO (Section 5.2.7) to four tabular datasets. The datasets we use are the four largest ones in terms of number of data points from a recent AutoGluon work (Erickson et al., 2020). For each of these datasets, 33% data are kept for testing, 25% are kept for validation and 42% are kept for training. These 4 datasets are:

1. **Covertype** from the UCI KDD archive (Hettich and Bay, 1999). This dataset aims to predict the forest cover type given cartographic variable input data. It contains 581,012 data-points, 54 input features and 7 classes.
2. **Airlines** from Elena Ikonomovska (Albert Bifet, 2009), the goal is to predict whether a given flight will be delayed or not given input data of the scheduled departure. It contains 539,383 data-points, 8 input features and 2 classes.
3. **Albert**. A competition dataset from the AutoML Challenge series (2015-2018) (Guyon et al., 2018). It contains 425,240 data-points, 79 input features and 2 classes.

4. **Dionis**. A competition dataset from the AutoML Challenge series (2015-2018) (Guyon et al., 2018). It contains 416,188 data-points, 61 input features and 355 classes.

The grammar we used is introduced in Section 5.2.7. And the baselines we used for comparison are aging evolution (AgE-1) (Real et al., 2019), Auto-Pytorch (Zimmer et al., 2020) and AutoGluon (Erickson et al., 2020). Aging evolution is introduced in Section 5.2.7. AutoGluon combines different supervised learning models such as neural networks, LightGBM, CatBoost, random forest, extra trees, and K-nearest neighbors, the hyperparameters of which are automatically tuned. In contrast, Auto-PyTorch only adopts neural network models but also uses an ensemble strategy to improve the accuracy. As for our approach, we use *AgEBO-NR-LR-BS* to indicate that the number of ranks for parallelization (NR), learning rate (LR) and batch size (BS) are also optimized using Bayesian optimization. The population size is fixed to 100 and the sample size is 10. More implementation details can be found in Egele et al. (2021) and the code can be found in the GitHub repo https://github.com/deephyper/NASBigData.

The comparison of AgE, AgEBO and Auto-Pytorch is shown in Figure 5.22. One can see that AgEBO achieves validation accuracy values that are higher than those of Auto-PyTorch within 30 minutes of search time. The observed differences in the accuracy values can be explained by two factors. First, Auto-PyTorch is not designed to generate a single neural network model but to generate multiple models and combine them using an ensemble strategy to have a good accuracy. Second, the architecture space of Auto-PyTorch is restricted to a smaller number of trainable parameters and smaller number of layers.

Table 5.1 shows the accuracy values of the best models and the corresponding inference time of AgEBO and AutoGluon. The table shows that the test accuracy values of AgEBO and AutoGluon are comparable on all four data sets. The accuracy improvement over AutoGluon is significant on Airlines and less significant on Covertype and Dionis. However, the key advantage stems from the inference time with the trained model. Given that AgEBO generates a single neural network model, the inference time is between 2.7 and 4.3 seconds. On the other

hand, AutoGluon relies on stacking a number of models, resulting in an inference time of about 7 minutes.

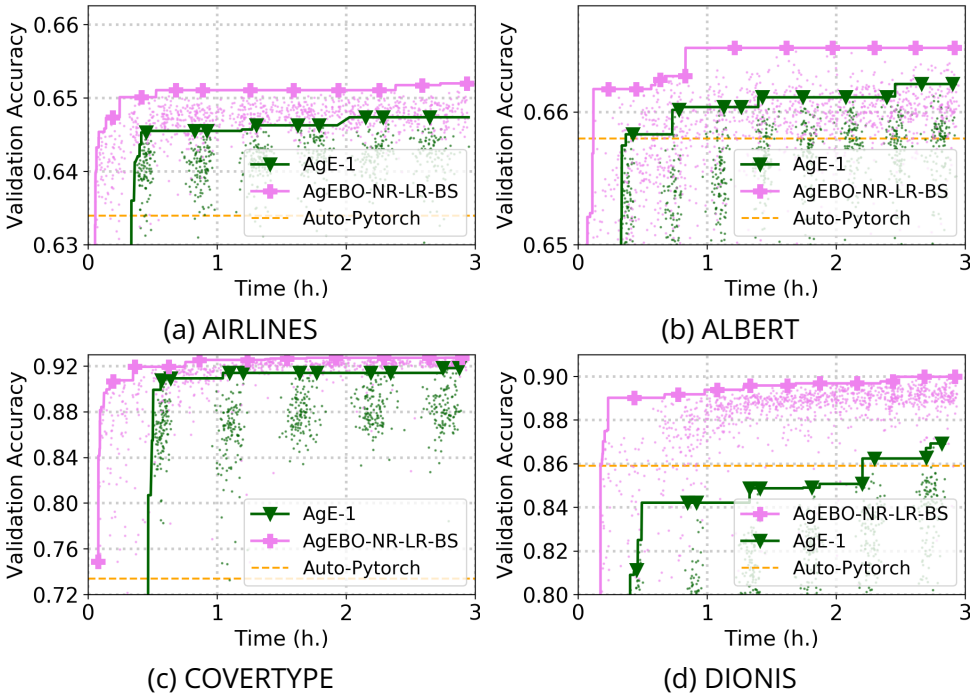


Figure 5.22 **Performances of AgEBO-NR-LS-BS on 4 large tabular datasets**: Airlines (5.22a), Albert (5.22b), Covertype (5.22c) and Dionis (5.22d). AgE-1 (Aging Evolution without data-parallelism) is the baseline experiment. A horizontal line shows the validation accuracy at the 20^{th} epoch of the model with the best validation accuracy found by Auto-Pytorch. Figures from (Egele et al., 2021).

5.4 . Discussion and Conclusion

5.4.1 . Grammars: Pros & Cons

Following recent popularity of NAS and AutoML in general, some recent works that also encode the search space of AutoML/NAS methods using formal grammars exist in the literature (Assunção et al., 2020; Estevez-Velarde et al., 2019; Katz et al., 2020). But these methods are far from being trending and remain a minority. In the following, we discuss some pros and cons of using grammars.

Some advantages of our approach using formal grammars are:

— **Generality.** As we discussed in the beginning of this chapter and showed with concrete examples later, context-free grammars are versatile enough to encode complex search spaces. Also, using grammars aligns very well with problems of algorithm search since the ultimate representation of algorithms is nothing but its *code*, written according to some grammar (think of C++ or Python). This means that the effective search space can be included in a context-free grammar search space;
— **Expressivity.** Context-free grammars are more expressive than regular grammars (or regular expressions), which in turn are more expressive than a finite search space;
— **Flexibility.** Once a grammar is given for a problem, the MCTS search strategy can be used straightforwardly without any modification. This is shown from the example results in previous section since we changed the grammar but used the same search strategy.
— **Explainability.** With grammars, one can analyze the learned syntax tree paths which are in the most cases comprehensive in a direct way. This gives edges to explainable solutions.

On the other hand, there are also some inconvenience of approaches using grammars.

— **Relatively high threshold.** The population familiar with formal grammars is relatively small. Formal grammars remain a domain reserved to computer scientists in general. Most importantly, the same grammar search space can be expressed in different ways, not all amenable to the same exploration. Hence designing grammars requires careful consideration. All these leave a relatively high threshold for applying grammars to real applications.
— **Learning difficulty.** When the grammar becomes complicated, many choices are involved and the number of possibilities grows exponentially (given the depth in the derivation tree). This raises challenges for the learning/search/optimization algorithms. This can be partially implied from the last example in the previous section. To overcome this one can think of meta-learning algorithms or algorithms that are capable of learning useful modules/bricks.

— **Unsolved problems.** During the experiments with the toy example in Appendix A, we noticed that if we change the rule

integer: digit integer

to

integer: integer digit

then the random generation method introduced in 5.2.5 (which we recall that the leftmost non-terminal is derived according to a distribution that favors rules with less non-terminal on the right hand side) tends to be never-ending, especially with small β. This means that even for grammars that encode the same search space, there may be very different search behavior. In the above case, this is related to the fact that we used *leftmost derivation*. Some possible ideas to remedy this is to consider restrictions on the size of the final expression produced. Ratle and Sebag (2000) made a very relevant investigation on this topic;

5.4.2 . Accelerating GramNAS Using Meta-learning

One main limitation of applying MCTS to achieve automatic neuronal architecture identification in the search space (architecture space) defined by a grammar – defining the GramNAS method proposed previously – is the slow convergence on image classification applications. The convergence is slow in particular compared to differentiable methods such as DARTS (Liu et al., 2019a).

Indeed, the evaluation of the leaf node (i.e. training a neural network from scratch) is quite time-consuming; however this is unavoidable if each architecture is evaluated without warm starting or weight sharing.

Let us focus on the MCTS part of the algorithm and discuss how to accelerate the identifiction of good subtrees.

Let us consider the simple case of two grammars:

$$\begin{aligned} G_1 &= (V, \Sigma, R_1, S) \\ G_2 &= (V, \Sigma, R_2, S) \end{aligned} \tag{5.11}$$

If the derivation rules in G_1 are included in those of G_2 ($R_1 \subseteq R_2$), it follows that the full tree of G_1 is included in that of G_2. Therefore the MCTS tree constructed with G_1 can be reused (the evaluations still hold,

assuming of course that one consider the same training dataset) and it can serve as warm start to conduct an MCTS process with G_2. This is analogous to the usage of pre-trained neural network in transfer learning.

As we have pointed out, the ultimate representation of algorithms is their code with a grammar. So the real power of using grammars comes in when one thinks of a universal encoding with a powerful grammar (think of any modern programming language that is Turing complete)

$$G^* = (V, \Sigma, R^*, S) \tag{5.12}$$

which encompasses grammars with which we have done search on $G_i = (V, \Sigma, R_i, S), i = 1, 2, 3, \ldots$ with

$$R_i \subseteq R^*, i = 1, 2, 3, \ldots \tag{5.13}$$

Formally, we can embed all learned MCTS trees of all G_i's into that of G^*, and reuse their statistics ($\hat{\mu}_i, n, n_i$, see Chapter 5). The real difficulty comes from finding an appropriate schedule: how to order the grammars G_i's and when to switch from G_i to G_{i+1}. At an abstract level, we face the same issue as an MCTS where each node corresponds to a *Many-Armed Bandit*, entailing a risk of over-exploration (Teytaud et al., 2007; Wang et al., 2008). In practice, this issue can be handled using a Progressive Widening strategy, gradually increasing the number of child nodes allowed in a visited node depending on the number of times it is visited Auger et al. (2013).

Another perspective for further work is to integrate a task representation in the above gradual GramNAS approach, supporting e.g. the initialisation of the MCTS tree depending on the problems archive and the representation of the current task.

This direction of research can take inspiration from similar ideas related to transfer learning and more generally meta-learning, currently explored in the communities of genetic programming (Dinh et al., 2015) and probabilistic context-free grammar (PCFG) (Han et al., 2020).

5.4.3 . Conclusion

We introduced a novel NAS framework combining formal grammars and search strategies such as Monte-Carlo Tree Search and evolutionary algorithms. We showed the generality and expressivity of context-free grammars for encoding search spaces and give some concrete examples for popular search spaces in the literature. The combination of grammar and MCTS was shown to be effective, on both the toy example in Appendix A and NAS problems. For the NAS-Bench-101 example, we achieved an accuracy of 94% on CIFAR-10 dataset, similar to state-of-the-art approaches such as aging evolution (Real et al., 2019). Our proposed GramNAS-AgEBO algorithm is proven effective and robust, and is competitive to some state-of-the-art approaches, at least when applied on tabular datasets. The search space (defined in Figure 5.14 and Figure 5.15) mainly consists of fully connected neural networks with skip-connections and would possibly show less advantages against convolutional neural networks when applied on for example image tasks. In terms of the usage of grammar, we discovered some unsolved problems concerning the derivation and observed the difficulty of learning when the grammar becomes more and more complicated. Finally, we drew attention from the research community of applying meta-learning for more efficient searching algorithms with grammars.

Algorithm 1: AgE (black) and AgEBO (black + blue)

inputs : P: population size, S: sample size, W: workers
output: highest-accuracy model in *history*

```
   /* Initialization                                                    */
1  population ← create_queue(P) // Alloc empty Q of size P
2  optimizer ← optimizer()
3  for i ← 1 to W do
4  |  model.h_m ← random_point(H_m)
5  |  model.h_a ← random_point(H_a)
6  |  submit_evaluation(model) // Not blocking
7  end
   /* Main loop                                                         */
8  while not done do
   |  // Query results
9  |  results ← get_finished_evaluations ()
10 |  if |results| > 0 then
11 |  |  population.push(results) // Aging population
   |  |  // Generate hyperparameter configs
12 |  |  optimizer.tell(results.h_a, results.m)
13 |  |  next ← optimizer.ask(|results|)
   |  |  // Generate model hyperparameter configs
14 |  |  for i ← 1 to |results| do
15 |  |  |  if |population| = P then
16 |  |  |  |  sample ← random_sample(population,S)
17 |  |  |  |  parent ← select_parent(sample)
18 |  |  |  |  child.h_m ← mutate(parent.h_m)
19 |  |  |  else
20 |  |  |  |  child.h_m ← random_point(H_m)
21 |  |  |  end
22 |  |  |  child.h_a ← next[i].h_a
23 |  |  |  submit_evaluation(child) // Not blocking
24 |  |  end
25 |  end
26 end
```

	AgEBO-NR-LR-BS		AutoGluon	
dataset	**Test Accuracy**	**Timing**	**Test Accuracy**	**Timing**
Airlines	**0.652** $\pm$ **0.002**	3.1	0.641	1124.9
Albert	0.661 $\pm$ 0.001	2.7	**0.688**	409.3
Covertype	**0.963** $\pm$ **0.001**	4.3	0.961	906.6
Dionis	**0.915** $\pm$ **0.0005**	3.2	0.907	1900.5

Table 5.1 **Evaluation of predictions with the best model from AgEBO** with optimized learning rate (LR), batch size (BS) and number of ranks (NR) against AutoGluon (Erickson et al., 2020) The table summarizes the test accuracy of the model on the same test sets as well as the time (seconds) taken to make these predictions on the same hardware also referred as inference time. Errors bars are obtained by training the best architecture from scratch for 5 times. Table from (Egele et al., 2021).

6 - Meta-learning

Meta-learning (Brazdil et al., 2008; Vanschoren, 2018) aims at leveraging past learning experiences to improve or accelerate the learning process for future tasks. Inspired by Sun-Hosoya (2019) and Rakotoarison et al. (2019), this chapter first presents a taxonomy of meta-learning *solutions* (complementing the categorization of meta-learning *problems* in Chapter 3) in terms of reinforcement learning, to situate the position of the problem. The international challenge MetaDL, organized to benchmark Few-Shot learning algorithms, is then presented and discussed, together with a real-world case study devoted to power systems. Lastly, some initial theoretical results for *zero-order meta-learning* (defined in Chapter 3) are presented.

6.1 . Reinforcement Learning Formulation of Meta-learning

To gain a global view of meta-learning, we first formulate the meta-learning problem in the setting of reinforcement learning (RL), inspired by the approaches of (Sun-Hosoya, 2019) and (Rakotoarison et al., 2019). We will see that this allows us to create a comprehensive taxonomy of meta-learning.

6.1.1 . The Reinforcement Learning Problem

We recall the classical setting of RL following (Sutton and Barto, 2018), as shown in Figure 6.1. An *agent-environment interface* defines the dynamics and the interactions. The *agent* can learn and make decisions and the *environment* can give feedback such as the *reward* and the *state* information to the agent. Formally, all possible states of the agent form the *state space* $\mathscr{S}$ and all possible actions of the agent form the *action space* $\mathscr{A}$. At each time step $t = 0, 1, 2, \ldots$, the agent receives the state info $s_t \in \mathscr{S}$ and can choose an action $a_t \in \mathscr{A}$ (or $a_t \in \mathscr{A}(s_t)$ that is specific to the current state s_t, with $\mathscr{A}(s_t) \subseteq \mathscr{A}$). Then one time step later, the agent receives a numerical reward $r_{t+1} \in \mathbb{R}$ and finds itself in a new state $s_{t+1} \in \mathscr{S}$.

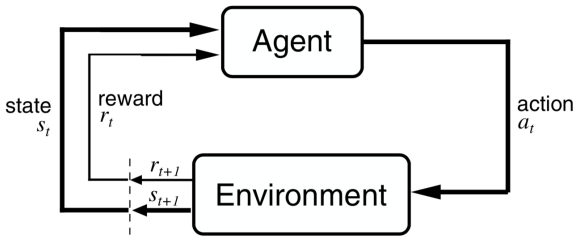


Figure 6.1 **Reinforcement Learning** with the *agent-environment interface*. Figure from (Sutton and Barto, 2018).

The goal is to find a *policy* $\pi : S \times A \to [0,1]$ that indicates the probability $\pi(s,a)$ of choosing action a while in state s. This policy should maximize the cumulative *expected return*:

$$R_t = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right] \tag{6.1}$$

where the r_t is obtained by applying the policy π at each step. The *discount rate* $\gamma \in [0,1]$ is a pre-defined parameter. Reinforcement learning problems are most often formalized within the *Markov decision process* (MDP) framework Sutton and Barto (2018), where the future only depends on the current state and current action. Sometimes the agent does not have access to the full *information* of the state variable s_t and can only make an observation o_t that only partially reveals s_t. Then we are in the case of *partially observable Markov decision process* (POMDP) (Åström, 1965) and this is the case for many games such as Battleship, Texas hold'em poker (Brown and Sandholm, 2017), Starcraft (Arulkumaran et al., 2019) and many others. These games are also called *imperfect-information games*.

The above reinforcement learning problem is very general and can be applied to countless scenarios and plays a very important role in artificial intelligence in general. In addition, we find it appropriate to indicate a correspondence between several fields of study. The notions of *state* (or *system* and environment), *action*, *reward* and *information* (SARI for all above) can be found in not only in RL, but also in optimization, game theory, machine learning in general and algorithm selection. For example, the PAPI (Rasmusen, 2005) (for **P**layer, **A**ction, **P**ayoff,

Information) formulation in game theory corresponds well to the SARI formulation in reinforcement learning. Other similar formulations also exist such as PEAS (Russell and Norvig, 2002) (for **P**erformance measure, **E**vironment, **A**ctuators and **S**ensors) in the artificial intelligence community. The correspondence between the different concepts used in the different disciplines is indicated in Table 6.1.

	Optimization	Reinforcement Learning (SARI)	Game Theory (PAPI)	Supervised Learning	Algorithm Selection	Artificial Intelligence (PEAS)
S	Search Space	System / World / Environment / State Space	Players (PAPI)	Task / Dataset / Generative Model	Task	Environment
A	Algorithm Step	Agent Action	Action	Learning Machine Output (α, β or γ)	Algorithm	Actuators
R	Risk / Objective Change / Loss	Reward / Return / Regret	Payoff / Utility	Risk / Loss / Score / Cost function / Metric / Performance	Performance / Score	Performance measure
I	Information / Latent variables / missing data	Information	Information	Information / Latent variables	Information	Sensors

Table 6.1 **SARI/PAPI** terminology across several fields. Many notions are very similar and/or closely related. However, some correspondence relationships (such as the 'S' and 'A' in optimization and RL) are rather loose and are not equivalent in a rigorous way. Note that the row 'I' for information is absorbed into the row 'S' in the RL formulation of Sutton and Barto (2018) but can be relevant in the context of partially observable Markov decision process (POMDP) (Åström, 1965).

6.1.2 . RL notions in Meta-learning

We now translate the very general reinforcement learning notions in the context of meta-learning. We recall from Chapter 3 that in meta-learning, we wish to find a γ-level algorithm that takes a meta-training set of past learning experiences as input and output a learning algorithm (or those with HPO), i.e. a β-level algorithm. For convenience, we recall some notations. The past experience is described by

$$\mathfrak{D} = \{(T_j, \beta_j, R_j)\}_{j=1}^{N} \in \mathscr{D} \tag{6.2}$$

where the T_j's are learning tasks and R_j are β-level algorithms (i.e. learning algorithms, HPO, etc). And we wish to construct a *meta-learning* algorithm $\gamma : \mathscr{D} \to \mathscr{B}$ that *learns* a β-level algorithm $\beta = \gamma(\mathfrak{D}) \in \mathscr{B}$ by exploiting past experience in $\mathfrak{D} \in \mathscr{D}$.

The objective of meta-learning problem (and more generally γ-level problem) is

$$\boxed{\begin{array}{l} \gamma^* = \arg\min_{\gamma} \frac{1}{|\mathfrak{D}_{te}|} \sum_{T \in \mathfrak{D}_{te}} R(\hat{\hat{\alpha}}; D_{te}) \\ \text{where } \hat{\hat{\alpha}} = \hat{\beta}(T) \text{ and } \hat{\beta} = \gamma(\mathfrak{D}). \end{array}} \tag{6.3}$$

Formulating the meta-learning problem as trying to find a γ seems very simple at first sight, but within this γ, there could be very complex dynamics. Very often, there is an *estimation* of the reward (typically the averaged accuracy) and *iterations* of adjusting and learning. And in the course of these iterations, RL notions naturally arise.

Example with a Meta-learning Competition

To show the above observation more concretely, we consider a meta-learning competition (similar to MetaDL challenge that we will introduce later in this chapter) as an illustrative example. This will also help to understand the context of the section on MetaDL challenge. In the considered meta-learning competition, a meta-dataset (i.e. meta-training set) is provided as *input data* and participants need to implement a meta-learner `z` (as shown in Figure 6.2) that can meta-learn with the method `meta_fit`, then learn with the method `fit` and predict with the method `predict`. An *ingestion program*, implemented by the orga-

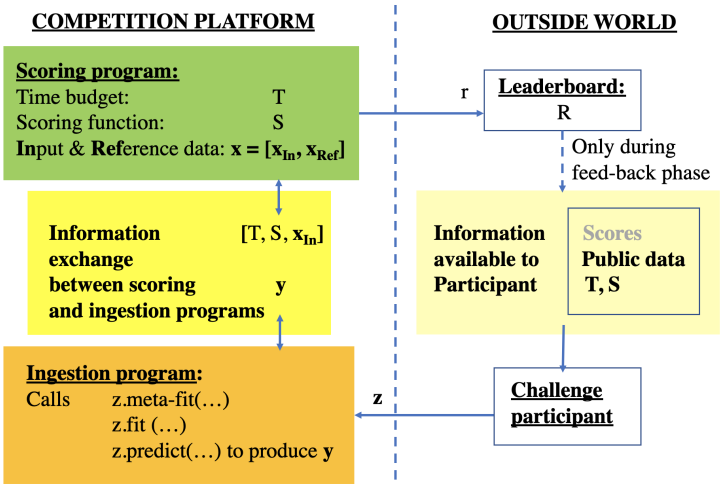


Figure 6.2 **Interaction of ingestion program and scoring program in a meta-learning competition.** A meta-dataset (i.e. meta-training set) is provided as *input data* and participants need to implement a meta-learner z that can meta-learn with the method `meta_fit`, then learn with the method `fit` and predict with the method `predict`. An *ingestion program* takes care of meta-learning, learning and prediction. Then a *scoring program* takes the predictions made by ingestion program, compares them to the ground truth (*reference data*) and returns the accuracy as performance, to be shown on the leaderboard.

nizer of the competition, serves to call these methods on input data. In other words, ingestion program takes care of all following three steps: meta-learning, learning and prediction. Then a *scoring program*, also implemented by the organizer, takes the predictions made by ingestion program, compares them to the ground truth (named *reference data*) and returns the accuracy as performance, which is to be shown on the leaderboard. This is in general how a competition with code submission works. To understand that RL notions naturally arise, we look into the inner part of the ingestion program, as shown in Figure 6.3. Since the agent (i.e. the z implemented by the participant) does not know the ground truth, it often needs a *surrogate scoring program* (in the same idea as Rice's *performance model* (Rice, 1976) as stated in Chapter 2) for estimating the actually outcome performance in the final evaluation. For any potential meta-learner $\_z$, this surrogate scoring program esti-

mates its performance which can then be considered as *reward*. Note that although this reward serves as an *estimation* of the true reward on the test set, it can still be considered as a 'true' reward, but on a validation set instead. This convention is adopted by many works in the AutoML literature (Liu et al., 2019a; Rakotoarison et al., 2019; Zoph and Le, 2016).

This choice of $\_z$ is then an *action*. And lastly, all records of past performances can be defined as *states*. Then this inner iteration in Figure 6.3 can be run to adjust the choice of $\_z$ and learn little by little.

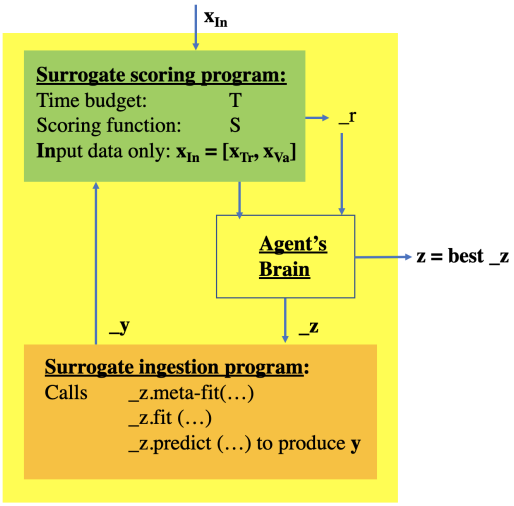


Figure 6.3 **Iterations inside the ingestion program in Figure 6.2.**

We note that this example only gives one possible way that RL notions arise in meta-learning.

In the following, we will discuss RL notions one by one in the context of meta-learning.

State Space

For us, the state space for meta-learning is defined to be the current state of the (meta-)dataset of past learning experiences (i.e. the meta-training set). According to the definition of meta-learning problem (γ-level problem) in the three-level formulation, the initial state is thus given by

$$s_0 := \mathfrak{D} = \{(T_j, \beta_j, R_j)\}_{j=1}^{N}$$

identical to that in Equation (6.2). As the agent can try different algorithms on different tasks/datasets, this meta-training set $\mathfrak{D}$ can be enriched iteratively (however we still consider the input of a meta-learning algorithm γ to be $s_0 = \mathfrak{D}$).

Although we wrote $\mathfrak{D} = \{(T_j, \beta_j, R_j)\}_{j=1}^N$, the objects T_j, R_j remain somewhat abstract and their concrete representation (e.g. data format) can vary from scenario to scenario. In some cases, they are only indexed by their ID and no further details are provided. Sometimes some meta-features (or statistics) on the algorithm R_j and/or on the task T_j are pre-computed and recorded. In others, full information (such as the code or the data) on R_j and task T_j is recorded (thus one can retrieve their exact definition and re-run any algorithms on any datasets as we want). And this observation exactly corresponds to what we introduced in Chapter 3 on the 3 classes of meta-learning problems and for convenience, we associate a tag for each of them:

1. **S0**: Zero-order meta-learning;
2. **S1**: First-order meta-learning;
3. **S2**: Second-order meta-learning;

Here the letter 'S' represents 'state'.

Action Space

The state space being defined, we now discuss what the actions can be in meta-learning. Actually what most meta-learning approaches in the literature do is to try an algorithm on a task and observe the outcome performance. So the agent is basically enriching the meta-dataset $\mathfrak{D} = \{(T_j, \beta_j, R_j)\}_{j=1}^N$. This observation is also shared by Sun-Hosoya (2019), who defined a form of meta-learning as what is called a *REVEAL* game. In this REVEAL game, the agent iteratively reveals information from executing an algorithm on a task.

Although in most cases, the algorithm chosen at each step for evaluation is a β-level algorithm, it can be the case where the agent directly chooses an α for the given task (and in this case only the validation set is required for evaluation and no training is needed). We shall see an example in the next section on our proposed approach on LEAP nets.

According to above discussion, we simply distinguish two cases for a meta-learning agent:

1. **A0**: choose an α;
2. **A1**: choose a β;

We will show examples later in Table 6.2.

Reward and Return

In meta-learning, we often have multiple objectives, thus several quantities can be considered as reward (and we recall that the return is the cumulative reward, possibly discounted). There is always the accuracy of classification in consideration but meta-learning often tries to optimize other aspects without losing much (or at all) on accuracy. These aspects include the number of examples (i.e. the amount of data) and the computational resources:

1. **RP**: Performance / accuracy
2. **RE**: number of examples
3. **RC**: computational resources

If one wishes to obtain similar accuracy but with very few examples (i.e. we are in the *few-shot learning* setting), one can use the notation for this type of reward by

$$RE|RP,RC$$

meaning that we want to optimize (minimize in this case) required RE (number of examples) while *fixing* similar RP (accuracy) and RC (computational resources). Sometimes we wish to minimize the required computational resources (for example in this case of mobile devices), then we can use the notation

$$RC|RP,RE$$

This is typically the case in *any-time learning* as in the AutoDL challenges (Liu et al., 2021) introduced in Chapter 4, where algorithms are required to make good predictions even when time budget is small.

We note that to compute the cumulative reward (i.e. the return), the *horizon*, that is the number of total steps, depends on the number

of tasks and algorithms we consider. As we always consider a finite number of algorithms and tasks, we are in an *episodic* RL setting (Sutton and Barto, 2018) rather than in a continuous setting.

Lastly, it is also possible to tackle a multi-objective optimization problem by considering all three objectives RP, RE, RC at the same time. Multi-objective RL approaches (Drugan and Nowe, 2013; Perez et al., 2014; Wang and Sebag, 2012) can thus be applied in this case.

Information

The notion of *information* in meta-learning is closely related to the notion of *state*. Actually the notion of information formulated in Sutton and Barto (2018) is completely absorbed in the notion of state, as we pointed out earlier. Thus in most cases, we can ignore the distinction between these two notion and always consider the *state*. However, we still list several cases in terms of information for the reader to gain some insights.

Without loss of generality, we suppose we are in the case of second-order meta-learning (since other cases can be considered as this case with different information), where a (data-)set of tasks (or a generative model of tasks) and a (data-)set of algorithms are considered. Then we can have following non-exhaustive list of scenarios:

1. No meta-features/statistics are recorded. Just partial data are given and meta-features are considered as unknown or latent variables. This is the case of AutoML challenges (Guyon et al., 2015, 2018) when the meta-features are not revealed to the participants. This is also the case in some scenarios related finance (e.g. Numerai(https://numer.ai/));
2. We know the summary statistic or generative model of the tasks. This is the case of LEAP net that we will introduce in Section 6.3;
3. We recorded meta-features of all past tasks. For example, consider data generated in some chemical experiments, then the data of the operator, the weather and the temperature are meta-features of these tasks. In the case of AutoDL challenges (see Chapter 4), the source, shape, number of examples, etc., are all

meta-features that we can record. It makes differences whether one has access to the above information.

Engineer and the Oracle

Here we briefly point out the fact that besides the agent-environment interface we mentioned above, some important external factors also exist. Typically, two of these external factors are the *engineer* who implements the algorithm (α, β or γ) and the *oracle* who knows the ground truth and makes the final judge/evaluation of the algorithm. In the context of a competition, the engineer is typically any participant and the oracle is the competition organizer. In most cases, the above two external factors have access to information that remains hidden to the agent (i.e. the implemented algorithm). The engineer can read the description of the competition, understand the semantic of the data and judge whether the competition is really useful (for his/her career or for the society). The organizer knows about the ground truth and maybe more information on the used datasets. These external factors are often neglected in academic discussions but they are the essential actors behind everything. Also, the whole point of AutoML is actually reducing the burden of the engineer. And each time when one jumps to one level up (e.g. from α-level to β-level, or from β-level to γ-level), one passes the job that was originally done by the engineer/human to the implemented agent. The higher level we are in, the more automation we achieve.

Although a further discussion on this topic is not in the scope of this thesis, we should keep the existence of these external factors (the engineer and the oracle) in mind.

Classification of Existing Works

Until now, we characterize meta-learning in turns of its SARI (State, Action, Reward, Information) which enabled us to characterize papers from the literature (see Table 6.2), and this allowed us also to position our own contributions. In what follows, we illustrate three cases in Section 6.2, 6.3 and 6.4.

6.2 . MetaDL Challenge

Although much research has been done in meta-learning (Brazdil et al., 2008; Finn et al., 2017; Vanschoren, 2018; Vanschoren et al., 2014; Vilalta and Drissi, 2002), it is still hard to fairly benchmark existing meta-learning approaches. During this thesis, we organize a meta-learning competition which will serve as an open and fair platform for evaluating meta-learning approaches and inspiring community efforts to advance this domain. In this competition, we choose to focus on a specific sub-field of meta-learning: **few-shot learning** (FSL). Following the taxonomy of the previous section, FSL corresponds to the $RE|RP,RC$ category in terms of the reward, which minimizes the need of traning examples while maintaining similar computational resources and performance (classification accuracy). And for state and action, the general setting of this competition corresponds to S2 and A1 respectively. Thus the SAR tags are $S2, A1, RE|RP,RC$.

Among diverse applications of few-shot learning, few-shot image classification has been receiving a lot of attention lately (Finn et al., 2017; Ravi and Larochelle, 2016; Snell et al., 2017; Sung et al., 2018), see Wang et al. (2020) for a recent survey). A lot of modern applications benefit from improving our understanding and results of such techniques. A simple application would be to tag photos on Facebook based on very few labels provided by the user. Also, one powerful feature of working with images is our natural ability to assess if an algorithm is performing well for the right reasons using tools such as GradCam (Selvaraju et al., 2017) (Gradient-weighted Class Activation Mapping) that can explain why a convolutional neural network makes a decision, therefore providing an extremely valuable resource for post-challenge analysis.

In order to fairly benchmark existing few-shot learning approaches and push forward the state of the art, we organized a competition in meta-learning dubbed *MetaDL challenge*, which is still on-going by the time of September 2021. This challenge is the first one to tackle the particular problem of few-shot learning for image classification. The crucial part of the challenge organization is to provide participants an easy API to create their solutions, as well as a general API so that the community can reuse it for problems in related areas (e.g. coupling

FSL with NAS and also RL problems). Therefore, it is important to provide a flexible framework that allows them to submit any type of FSL algorithm. We took inspiration from the design of famous python packages such as the **Scikit-Learn** package (Pedregosa et al., 2011) and also the three-level formulation introduced in Chapter 3.

6.2.1 . Competition workflow

The competition is currently running online, which means that the participants need to submit a Python script (probably with other files) implementing their meta-learning algorithm. The competition will be hosted on a dedicated challenge platform CodaLab (noa) and the submissions will be executed on some virtual machines (supported by Microsoft Azure). The script will have to respect a specific API that we defined as the challenge organizer. This API is designed to be flexible to be used for describing any meta-learning procedure. And this is because we actually provide the possibility for *second-order meta-learning* (as described in Chapter 3) with all meta-training data provided. It defines 3 main classes for which their methods need to be overridden to define a meta-learning algorithm. Its design relies on the definition of the different levels of algorithms that have been described in Chapter 3. These 3 classes are [1]:

— A **MetaLearner** class: It has a **meta_fit()** method that encapsulates the meta-training procedure. Using the previously defined notation, it essentially process the meta-dataset and capture the reusable information across meta-training tasks. It takes a meta-training set $\mathfrak{D}_{tr}$ as an argument and outputs a **Learner** object (defined below). This corresponds to a meta-learner γ in the three-level formulation in Chapter 3.

— A **Learner** class: It has a **fit()** method that encapsulates the training procedure. It takes the training dataset D_{tr}^{i} as an argument from a task T_i along with the associated information from the meta-learning procedure to output a **Predictor** object. This corresponds to a learner β in the three-level formulation.

1. Code available at https://github.com/ebadrian/metadl

— A **Predictor** class: It has a **predict()** method that predicts the labels of test examples. It takes a test set D^i_{te} as an argument from the associated T_i. This corresponds to a predictor α (applied to a set of examples) in the three-level formulation.

Each participant has to override these methods to create their own meta-learning algorithm. We will discuss the implementation details in the results section.

The learning procedure

Once a participant submits an algorithm, the ingestion program (in Figure 6.2) feeds data in form of *episodes*. Each episode (forming a learning task) is defined by $T_i = (D^i_{tr}, L^i | P^i, D^i_{te})$ (here we associate an *unknown* test set D^i_{te} with unlabeled examples to the task) or simply

$$T_i = (D^i_{tr}, D^i_{te})$$

when there is no confusion on the other components. A typical task, referred to as N-*way* K-*shot* task, involves a training set with N classes and K examples (typically $N = 5, K = 1$)

$$N = 5, K = 1$$

but other settings can be easily integrated in CodaLab.

Meta-training step. The ideal few-shot learning algorithm is able to leverage the knowledge gathered from all prior tasks in $\mathfrak{D}_{tr}$ to increase performance on unseen tasks at meta-test time. The tasks are first sampled from the original tasks distribution, forming a $\mathfrak{D}_{tr}$. The way we sample these tasks will be described in the next sub-section data. Episodes are then fed to the Meta-Learner through the **meta_fit** method.

Meta-testing step. During this step, a pre-defined number of tasks would be sampled (e.g. 600 in the competition). Each of these tasks is also of the form $T_i = (D^i_{tr}, D^i_{te})$ and the submitted algorithm does not have access to the test labels. For each new task, we use the learner generated from the meta-training part, to output a predictor using D_{tr}. This process is encapsulated in the Learner's **fit** function. Then, we'd

predict the labels of test examples using the resulting predictor. The different meta-learning algorithms differ in the way they generate a Learner and how they use the information from $\mathfrak{D}_{tr}$ coupled with D_{tr}^i to perform well on D_{te}^i. This way, we measure the capacity to quickly adapt to a new task from just a few examples.

Evaluating performance. One crucial design choice is the way we assess an algorithm's performance. Performance is measured with the standard classification performance,

$$ACC = \frac{1}{|D_{te}|} \sum_{y \in D_{te}} \mathbb{1}_{\hat{y}=y} \tag{6.4}$$

with D_{te} the test set, where $\hat{y}$ and y are respectively the predicted label and the true label.

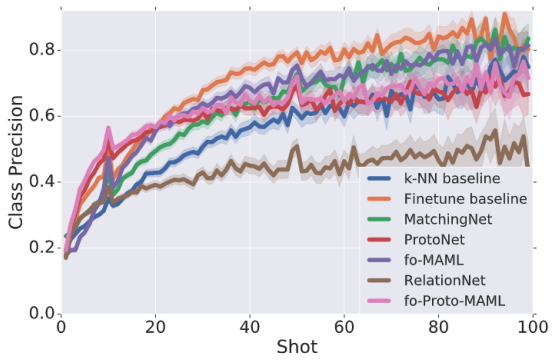
Few-shot learning algorithms perform differently under different regimes. More specifically, the rate at which they benefit from adding more examples in D_{tr} vary (c.f. Figure 6.4). We are interested in this particular insight developed in (Triantafillou et al., 2019), i.e. a way to measure an algorithm's performance across these different regimes. At the moment, we are considering the possibility to use a more sophisticated metric than the simple categorical accuracy by computing the area under learning curves in which the x-axis is denoting shots, while the y-axis could represent accuracy. As an example, the following figure taken from (Triantafillou et al., 2019) displays the accuracy for different algorithms across various regimes (i.e. different N and K).

6.2.2 . Data

Task generation

Tasks (or episodes) in this competition are generated as follows. Consider a dataset (for example the Omniglot (Lake et al., 2015) dataset) that contains a large number of classes. We divide the classes into 2 disjoint sets $\mathscr{C}_{tr}$ and $\mathscr{C}_{te}$. These sets represent "pools" from which classes we generate episodes for the meta-train and meta-test part respectively.

An episode is defined by selecting uniformly N=5 classes, and selecting $K = 1$ (respectively $K = 19$) examples in each selected class to form



(b) Shots analysis

Figure 6.4 **Shots analysis**, figure from (Triantafillou et al., 2019). Each curve represents a few-shot learning algorithm meta-trained on various datasets and evaluated only on ImageNet (Krizhevsky et al., 2012) tasks.

D_{tr}^i (resp. D_{te}^i). Indeed, Omniglot contains over 1600 classes of 20 examples each. This number essentially depends on how much examples are available in a dataset. Also it is possible for participants to use the type of data augmentation they want on the $\mathfrak{D}_{tr}$. It is worth noting that the sampling procedure is controlled on the challenge organizer's side.

In order to generate tasks/episodes in the meta-test set, we proceed in the same way with classes in $\mathscr{C}_{te}$ (without the data augmentation).

(Meta-)Datasets

We previously mentioned that tasks are generated using only one (meta-)dataset. This procedure could be extended with multiple datasets by simply adding the corresponding classes in $\mathscr{C}_{tr}$ and $\mathscr{C}_{te}$. The competition is divided into 3 different phases. In each phase, we construct a meta-training set $\mathfrak{D}_{tr}$ and a meta-test set $\mathfrak{D}_{te}$ as described above.

1. **Public phase**: This first phase consists of letting participants have full access to famous datasets in the few-shot learning literature. They can test their algorithms and analyse their results using directly the labeled examples.
2. **Feedback phase**: Participants can submit their code and have a performance feedback on the meta-test dataset associated to this phase.

3. **Final phase**: The performance on these private tasks are the one we use to rank participants at the end of the competition. The data we use for this phase are not public and are not available on the internet.

For public phase, we use the Omniglot dataset. We divided the classes from Omniglot dataset in two sets. Omniglot has 1622 classes and the assignment of classes in each of these set is identical to the original split of the authors. The number of classes in each classes set is as follows : $|\mathscr{C}_{tr}| = 1200$ and $|\mathscr{C}_{te}| = 422$. A sample of the Omniglot classes is displayed in Figure 6.5.

Figure 6.5 **A sample of classes taken from the Omniglot dataset.** (Lake et al., 2015)

Some meta-datasets we consider using in the feedback phase are:

— Omniglot dataset (Lake et al., 2015) : handwritten characters across 1623 classes.

— Caltech UCSD Birds 200 (Wah et al., 2011): 6033 images across 200 classes.

— A subset of Quick Draw (https://quickdraw.withgoogle.com/): 50 millions drawings across 375 classes.

— Hammer, Hippocrate which are medical image datasets from AutoDL challenges (see Chapter 4): respectively containing 7 classes, 10015 images and 2 classes, 220025 images.

— Ideal and Saturn datasets, which are aerial datasets from AutoDL challenges(see Chapter 4), respectively containing 45 classes, 31500 images and 3 classes, 405000 images

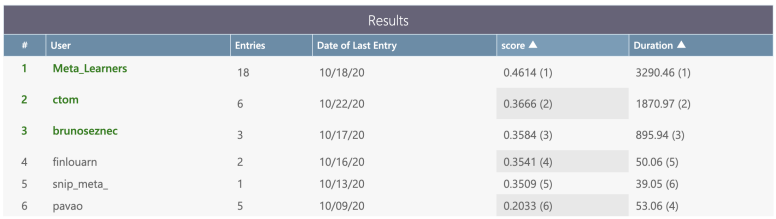
The choice of datasets and specifically the classes in $\mathscr{C}_{tr}$ and $\mathscr{C}_{te}$ from which we create our meta-training set and meta-test set is not the only important issue. The type of classes we consider together in a same meta-dataset should also be explored. Indeed, some datasets contain very subtle difference among some of their classes (fine-grained image classification) while others are more coarse. Often, the meta-learning algorithm performances would be drastically different if we meta-train algorithms with the former type of classes and evaluate on the latter type of classes. It is our responsibility as challenge organizers to design the selection of classes in order to gain as precise and wide knowledge about the algorithms capabilities as possible. These considerations are also discussed in Triantafillou et al. (2019) and we will continue to investigate these concerns.

6.2.3 . Baseline Methods

We provide two baseline methods to the participants. The first baseline is the first order approximation of MAML algorithm (fo-MAML) (Finn et al., 2017) (along with its original version). The second baseline is *Prototypical Networks* (Snell et al., 2017), which was not yet ready for generating baseline results when writing. Thus for comparison, we adopted a naïve baseline of a fully connected neural network (Fully Connected) with no hidden layer and with no meta-learning at all. Along with the implementation of baselines as well as their testing script to perform crucial sanity checks, we also implemented scripts to make any datasets that was used in a previous AutoDL competition to be usable in the data generation pipeline format presented in (Triantafillou et al., 2019).

In the next section, we present the results we obtained by meta-training our implementation of the fo-MAML algorithm on the Omniglot dataset (Lake et al., 2015).

6.2.4 . Baseline Results



Results

#	User	Entries	Date of Last Entry	score ▲	Duration ▲
1	Meta_Learners	18	10/18/20	0.4614 (1)	3290.46 (1)
2	ctom	6	10/22/20	0.3666 (2)	1870.97 (2)
3	brunoseznec	3	10/17/20	0.3584 (3)	895.94 (3)
4	finlouarn	2	10/16/20	0.3541 (4)	50.06 (5)
5	snip_meta_	1	10/13/20	0.3509 (5)	39.05 (6)
6	pavao	5	10/09/20	0.2033 (6)	53.06 (4)

Figure 6.6 **Leaderboard of MetaDL challenge** by the time of Oct 2020.

We meta-trained the algorithm on $\mathfrak{D}_{tr}$ constructed with $\mathscr{C}_{tr}$ and performed the evaluation on 600 episodes generated with the classes in $\mathscr{C}_{te}$. The meta-training part outputs a Learner, which is a neural network with specific *initial* weights θ^*, which allows to fine-tune the model with D^i_{tr} and evaluate on D^i_{te} for each unseen task/episode T_i. The fine-tuning part is performed with the Stochastic Gradient Descent (SGD) algorithm.

Table 6.3 shows the results from our fo-MAML implementation. The different learning rate values are presented in the first column. The second column represents the average categorical accuracy across 600 episodes along with its standard deviation. The third column represent the accuracy of the same neural network (architecture-wise) with randomly initialized weights that were trained with D^i_{tr} of each task. We see that small changes of hyperparameters (learning rate in this case) can drastically improve the performance of this algorithm up to 99% accuracy across tasks and won't be discussed here. However, we can already notice the meta-learning effect by comparing the results between the fo-MAML and Fully Connected implementation.

We meta-trained our meta-learning algorithms on one Microsoft Azure machine with one Nvidia Tesla M60 GPU for about 7 hours. From the result, we see that meta-learning extremely helps to improve the accuracy in the few-shot setting indeed.

By the time of September 2021, the MetaDL for NeurIPS 2021 is still on-going via the website https://autodl.lri.fr/competitions/210. We show the state of the leaderboard of MetaDL for AAAI 2020 in Figure 6.6

6.3 . LEAP nets and Super-generalization

In this section, we introduce an application of a special form of meta-learning. It concerns a type of *transfer learning* (Pan and Yang, 2010) that is capable of achieving *super-generalization* which we will define in Section 6.3.2.

The application domain will be power systems with a grid transporting electricity. The goal here is to develop a neural network model of the power grid, which allows us to replace *computationally costly simulators using first principles of physics* in the computation of power flows, following other prior work (Donnot et al., 2018a,b; Hossen et al., 2017; Nguyen, 1995). Following the taxonomy in Section 6.1, this approach corresponds to the SAR tags

$$S1, A0, RC|RP, RE.$$

This work is a collaboration with Benjamin Donnot, Balthazar Donon and others. One paper on this topic is published in Elsevier's Neurocomputing 2020 (Donon et al., 2020a).

6.3.1 . The Problem Setting with a Power Grid

Figure 6.7 illustrates the problem setting on a small power grid. A line going over its thermal limit will be put out of service. Hence, the grid must be reconfigured quickly to re-balance current flows before that happens. Although many types of actions/reconfigurations can be considered, we limit ourselves to one particular type of action: reconfiguration of the grid by *splitting* or *merging* nodes at substations. The figure shows a solution consisting of a node splitting in a substation. We call that a grid topology change. The space of possible grid topologies grows exponentially with the number of substations (nodes in the grid). For example, the French high-voltage transmission grid includes $N \approx 6200$ substations, with more than a dozen possible configurations per substation and thus $\gtrapprox 10^N$ possible grid topologies. Even if only a small number of those are achievable, the search space is still humongous. Our challenge is to devise a neural architecture and a training method such that, using a training dataset that contains only a few grid topologies, power flows can still be *accurately predicted* for topologies *never seen before*.

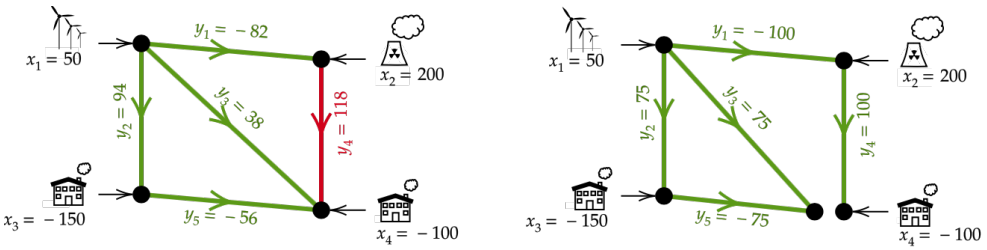


Figure 6.7 **Problem setting for the LEAP nets.** Electricity is transported from production nodes (top) to consumption nodes (bottom), through lines (green and red edges) connected at substations (black circles), forming a transmission *grid* of a given topology $\boldsymbol{\tau}$. Injections $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ (production or consumption) add up to zero. Grid operators (a.k.a. *dispatchers*) should maintain current flows $\boldsymbol{y} = S(\boldsymbol{x}; \boldsymbol{\tau})$ below thermal limits. Left: Line $\boldsymbol{y}_4$ goes over its thermal limit 100. Right: A change in topology (splitting of bottom right node) brings $\boldsymbol{y}_4$ back to its thermal limit.

Our approach uses a neural network

$$\hat{\boldsymbol{y}} = NN(\boldsymbol{x}; \boldsymbol{\tau}) \tag{6.5}$$

to emulate the power grid

$$\boldsymbol{y} = S(\boldsymbol{x}; \boldsymbol{\tau}) \tag{6.6}$$

(do not mix up this S with the S as a selection mapping in Chapter 3) in which the inputs $\boldsymbol{x}$ are so-called "injections" (productions and consumptions) and the outputs $\boldsymbol{y}$ are the power flows on all the connecting lines of the grid. The system is parameterized by structural parameter vector $\boldsymbol{\tau}$ encoding various grid topologies, which includes our *actionable parameters*. We use $\boldsymbol{\tau}$ to encode changes in our neural network architecture.

6.3.2 . Proposed LEAP Architecture

Our objective is to approximate a function $\boldsymbol{y} = S(\boldsymbol{x}, \boldsymbol{\tau})$ that maps input data $\boldsymbol{x}$ (*e.g.* power production and consumption) to output data $\boldsymbol{y}$ (*e.g.* power flows), parameterized by a discrete "grid topology vector" $\boldsymbol{\tau}$, taking values in an action space (all possible power-grid topologies *e.g.* line interconnections). For any fixed topology $\boldsymbol{\tau}$, training data pairs

$\{\boldsymbol{x}, \boldsymbol{y}\}$ are drawn *i.i.d.* according to an unknown probability distribution. In our application setting, $\boldsymbol{x}$ is drawn randomly, but $S(\boldsymbol{x}, \boldsymbol{\tau})$ is a deterministic function implementing Kirchhoff's circuit laws, calculated by a physical simulator that we wish to approximate.

We call *simple generalization* the capability of a neural net $\hat{\boldsymbol{y}} = NN(\boldsymbol{x}, \boldsymbol{\tau})$ to approximate $\boldsymbol{y} = S(\boldsymbol{x}, \boldsymbol{\tau})$ for test inputs $\boldsymbol{x}$ not pertaining to the training set, when $\boldsymbol{\tau}$ values are drawn *i.i.d.* from a distribution that remains the same in training and test data (this includes the case of a fixed $\boldsymbol{\tau}$). Conversely, if values of $\boldsymbol{\tau}$ are drawn according to a *source domain* distribution in training data and from a different *target domain* distribution in test data, then we will talk about *super-generalization*.

One particularity of our application domain in terms of transfer learning is that we have one *primary* "reference" source domain (corresponding in the power grid to a reference grid topology $\boldsymbol{\tau}^{\emptyset} = (0, 0, 0, \dots)$, around which *small* variations are made. This is a generic scenario in the industry for systems that operate around nominal conditions, thus we anticipate that our method could be extended to other similar situations. In our application setting, we can easily get a lot of training data in the reference topology (corresponding to the typical way in which the grid is operated). We have comparably very little data available for training from other *secondary* source domains, corresponding to unary changes in grid topology $\boldsymbol{\tau}^{i} = (0, 0, 1, \dots)$ (a single 1 at position i). Finally, we have extremely scarce data or no data at all available for training from domains corresponding to double changes $\boldsymbol{\tau}^{ij}$, or higher order changes (considered target domains). This motivates our architectural design.

Our proposed Latent Encoding of Atypical Perturbations network, or LEAP net (Figure 6.8), is composed of three parts: An Encoder $\boldsymbol{E}$, learning an embedding of the input data x; a Decoder $\boldsymbol{D}$, learning how to perform the required task within this latent representation; and a Latent module $\boldsymbol{L}_{\tau}$, placed between the $\boldsymbol{E}$ and $\boldsymbol{D}$ where $\boldsymbol{\tau}$ intervenes. The overall arhitecture is given by:

$$\boldsymbol{L}_{\tau} \quad : \quad h \quad \rightarrow \quad \boldsymbol{d}(\boldsymbol{e}(h) \odot \boldsymbol{\tau}) \tag{6.7}$$

$$\hat{y} \quad = \quad \boldsymbol{D} \circ (\boldsymbol{I} + \boldsymbol{L}_{\tau}) \circ \boldsymbol{E}(x) \tag{6.8}$$

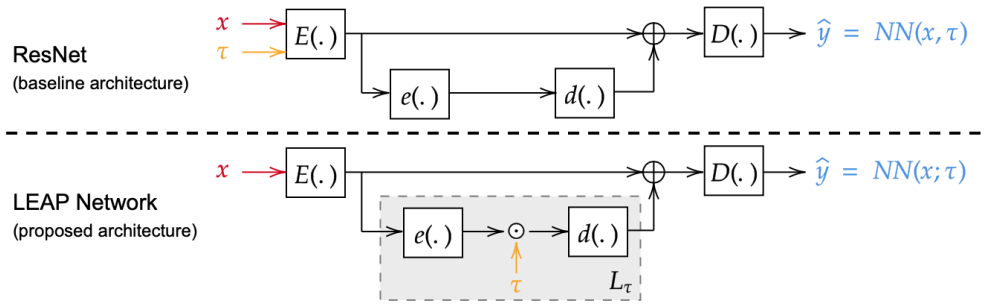


Figure 6.8 **Baseline and LEAP architecture:** Top: ResNet (He et al., 2016) architecture, with $\boldsymbol{\tau}$ as input. Bottom: Proposed LEAP net: $\boldsymbol{\tau}$ intervenes in the latent embedding space. The effect is to make a "leap" in latent space.

where $\boldsymbol{E}$ and $\boldsymbol{e}$ (encoders) and $\boldsymbol{D}$ and $\boldsymbol{d}$ (decoders) are all differentiable functions (typically implemented as artificial neural networks). The $\odot$ operation denotes the component-wise multiplication and $\circ$ the function composition. If the system is in the reference topology $\boldsymbol{\tau}^{\emptyset}$, predictions are made according to $\hat{\boldsymbol{y}} = \boldsymbol{D} \circ \boldsymbol{E}(\boldsymbol{x})$. A typical way in which we train LEAP nets is to use a lot of training data in the reference topology $\boldsymbol{\tau}^{\emptyset}$ (primary source domain), very few examples for each of the unary changes $\boldsymbol{\tau}^{i}$ (secondary source domains), and we wish the network to generalize to target domains corresponding to double $\boldsymbol{\tau}^{ij}$ or higher level changes.

While our architecture draws inspiration from both Dropout (Srivastava et al., 2014) and Residual Neural Networks (He et al., 2016), in its mathematical formulation, the underlying concept is quite different. Here we first embed x in a latent space by applying $\boldsymbol{E}(x)$. Then, based on $\boldsymbol{\tau}$ and the location of $\boldsymbol{E}(x)$ within the latent space, we compute the corresponding leap $\boldsymbol{L}_{\tau} \circ \boldsymbol{E}(x)$. Then we decode the signal by applying $\boldsymbol{D}$. Those latent leaps contain information about how much the system actually deviates from the reference state, and in which direction. Hence, our architecture only needs to learn to modulate the system response around its nominal value.

6.3.3 . Experimental Results

We present experimental results for our target application on simulated and real data. Synthetic data allows us to perform controlled systematic experiments and compare neural network approaches with

a standard baseline (DC approximation, in which one writes the vector of power injections as a linear function of the vector of voltage angles. Here 'DC' is for Direct Current) in power systems. Real data allows us to check whether our method scales computationally while providing prediction accuracies that are acceptable for our application domain.

Case 118 synthetic data benchmark

We conducted controlled experiments on a standard medium-size benchmark from "Matpower" (Zimmerman and et al., 2011), a library commonly used to test power system algorithms (Alsac and Stott, 1974): case118, a simplified version of the Californian power grid (dim $\boldsymbol{x}$ = 153 injections and dim $\boldsymbol{y}$ = 186 power lines). Topology changes consist in reconfiguring line connections in one or more substations (see Figure 6.7). Such changes are more complex than simple line disconnections considered in (Donnot et al., 2018a). There are $11\,558$ possible unary actions (corresponding to single node splitting or merging, compared to the reference topology). To build the Source domain training and test sets, we sampled randomly 100 $\boldsymbol{\tau}^{(i)} \in \mathscr{T}^{Source}$. In the reference topology ($\boldsymbol{\tau}^{\emptyset}$), we sampled 50000 input vectors $\boldsymbol{x}$. But for each $\boldsymbol{\tau}^{(i)}$, we sampled only 1000 input vectors $\boldsymbol{x}$. We used Hades2 [2] to compute the flows $\boldsymbol{y}$ in all cases. This resulted in a training set of $150\,000$ rows (each row being one triplet $(\boldsymbol{x}, \boldsymbol{\tau}^{(i)}, \boldsymbol{y})$). We created an independent test set of the same size in a similar manner.

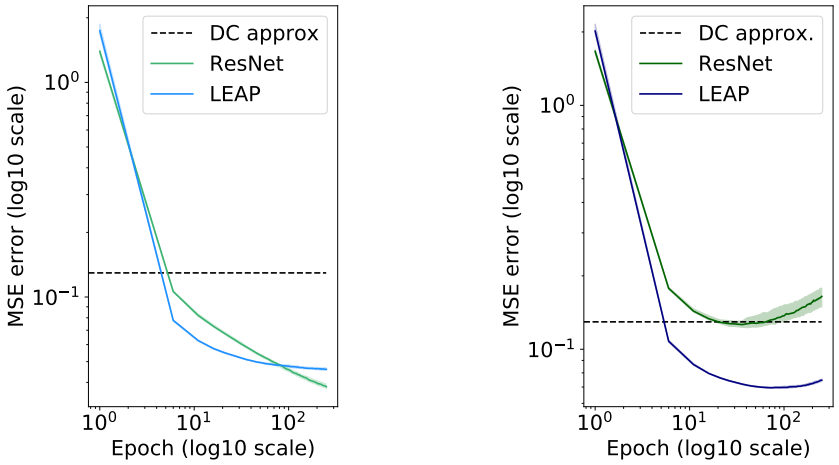
We proceeded differently for the Target dataset. We sampled 1500 (Target domains: $\boldsymbol{\tau}^{(ij)} \in \mathscr{T}^{Target}$) among the 4950 possible double actions $\boldsymbol{\tau}^{(ij)} = \boldsymbol{\tau}^{(i)} \vee \boldsymbol{\tau}^{(j)}$, $\boldsymbol{\tau}^{(i)}$ and $\boldsymbol{\tau}^{(j)} \in \mathscr{T}^{train}$. Then, for each of these 1500 $\boldsymbol{\tau}^{(ij)}$, we sampled 100 inputs $\boldsymbol{x}$ (with the same distribution as the one used for the training and regular test set). We used the same physical simulator to compute the $\boldsymbol{y}$ from the $\boldsymbol{x}$ and the $\boldsymbol{\tau}$. The super-generalization set counts then $150\,000$ rows, corresponding to $150\,000$ different triplets $(\boldsymbol{x}, \boldsymbol{\tau}^{(ij)}, \boldsymbol{y})$.

We compare the proposed LEAP net with two benchmarks: the DC approximation, a standard baseline in power systems, which is a linearization of the AC (Alternative Current) non-linear powerflow

2. Freeware available at http://www.rte.itesla-pst.org/.

equations, and the baseline neural network architecture (Figure 6.8) in which $\boldsymbol{\tau}$ is simply an input. The mean-square error was optimized using the Tensorflow Adam optimizer. To make the comparison least favorable to LEAP net, all hyperparameters (learning rates, number of units) were optimized by cross-validation for the baseline network.

Figure 6.9 indicates that the LEAP net (blue curves) performs better than the DC approximation (black line) both for regular and super generalization. Figure 6.9b shows that the baseline neural network architecture (green curve) is not viable: not only does it perform worse than the DC approximation, but its variance is quite high. While it is improving in regular generalization with the number of training epochs, its super-generalization performances get worse.

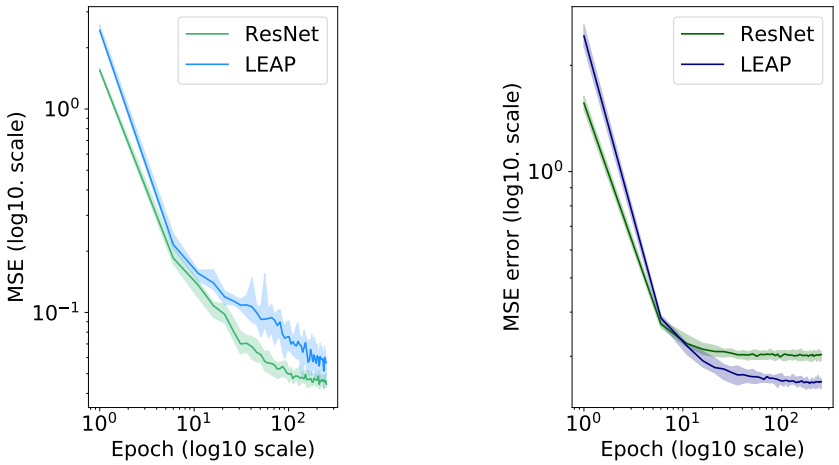


(a) Regular generalization (b) Super generalization

Figure 6.9 **Synthetic data (case 118).** Neural nets trained with 15000 injections, for $\boldsymbol{\tau}^{\emptyset}$ and unary changes $\boldsymbol{\tau}^{(i)}$. (a) **Regular generalization.** Test injections for **unary changes** $\boldsymbol{\tau}^{(i)}$. (b) **Super-generalization.** Test injections for **double changes** $\boldsymbol{\tau}^{(ij)}$. Error bars are [20%, 80%] intervals, computed over 30 repeat experiments.

Real French ultra-high voltage power grid data

We now present results on a part of the French ultra-high voltage power grid: the "Toulouse" area with 246 consumption nodes, 122



(a) Regular generalization

(b) Super generalization

Figure 6.10 **Real data from the ultra high voltage power grid.** The neural net in both cases is **trained from data until May 2017**. (a) **Regular generalization.** Test set made of randomly sampled data in same time period as training data. (b) **Super-generalization.** Test set made of the months of June and July 2017.

production nodes, 387 lines and 192 substations often split in a variable number of nodes. The inputs $\boldsymbol{x}$ representing injections (production and consumption) are of dim $\boldsymbol{x} = 368$) and the outputs $\boldsymbol{y}$ (flows) of dim $\boldsymbol{y} = 387$. In this study, $\boldsymbol{x}$ and $\boldsymbol{y}$ come from real historical data from the company RTE [3]. One important difference when using played-back data, compared to simulation, is that we cannot intervene (this is strictly observational data). To place ourselves in a realistic transfer learning setting, we used data from 2012 to May 2017 for $\mathcal{T}^{Source}$ and data from June and July 2017 for $\mathcal{T}^{Target}$. This favored changes in $\boldsymbol{\tau}$ distribution. Another key difference in real data is "actions space". In real data *actual* grid topologies (specifying line interconnections) are not precisely recorded. Only information on *line outages* is available to us as surrogate information on topology. This makes the neural net task harder: it must learn the effects of latent topological changes. This unfortunate loss of information on exact grid topology interventions makes it impossible for us to compare our method to the DC approximation: computing

3. Even in real records, flows are estimated, not measured.

this approximation requires a full description of the topology. The results of Figure 6.10 yield the same conclusions as in the previous section: the LEAP model generalizes not only to data drawn from a similar distribution it was trained on (Figure 6.10a) but also to unseen grid states (Figure 6.10b), better than the reference architecture, which is a critical property for our application.

6.3.4 . Theoretical Analysis of Super-generalization

In this section we formally prove the *super-generalization* ability of LEAP nets, when modeling systems with *additive perturbation*. It is important to note that LEAP nets are not limited to modeling additive perturbations. This simple theoretical analysis can be thought of as a "sanity check". In our experimental section we will show various empirical cases of super-generalization in our application setting of power system, not limited to additive perturbations.

A system with *additive perturbations* is defined as a system $S(\boldsymbol{x}, \boldsymbol{\tau})$ that satisfies

$$\begin{cases} S(\boldsymbol{x}, \boldsymbol{\tau}^{\emptyset}) = F(\boldsymbol{x}) \\ S(\boldsymbol{x}, \boldsymbol{\tau}^{i}) = F(\boldsymbol{x}) + \varepsilon_i(\boldsymbol{x}), \quad i = 1, \ldots, c \end{cases} \tag{6.9}$$

and

$$S(\boldsymbol{x}, \boldsymbol{\tau}^{\mathscr{I}}) = F(\boldsymbol{x}) + \sum_{i \in \mathscr{I}} \varepsilon_i(\boldsymbol{x}), \quad |\mathscr{I}| \geq 2 \tag{6.10}$$

for some (unknown) deterministic functions $F(\boldsymbol{x}), \varepsilon_1(\boldsymbol{x}), \ldots, \varepsilon_c(\boldsymbol{x})$.

We begin with a theorem showing the super-generalization ability of LEAP nets in the case where a trained model makes perfect predictions on data coming from the same distribution as the training data. Then we generalize this result to the noisy case with imperfect predictions.

Theorem 2. (Super-generalization) *Let $S(\boldsymbol{x}, \boldsymbol{\tau})$ be a system satisfying Equations 6.9 and 6.10 ; and let $NN(\boldsymbol{x}, \boldsymbol{\tau})$ be a LEAP net with* linear *decoders $\boldsymbol{d}$ and $\boldsymbol{D}$. If $NN(\boldsymbol{x}, \boldsymbol{\tau})$ is trained to make perfect predictions on* simple perturbations *(data triplets $(\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{y})$ coming from distribution defined by Equation 6.9), then it will make perfect predictions on* combinations of perturbations *(data coming from test distribution defined by Equation 6.10).*

Proof. We recall the LEAP Net architecture $NN(\boldsymbol{x}, \boldsymbol{\tau})$ which gives predictions

$$\hat{\boldsymbol{y}} = NN(\boldsymbol{x}, \boldsymbol{\tau}) = \boldsymbol{D}(\boldsymbol{E}(\boldsymbol{x}) + \boldsymbol{d}(\boldsymbol{e}(\boldsymbol{E}(\boldsymbol{x})) \odot \boldsymbol{\tau})) \in \mathbb{R}.$$

Since we assumed $l = 1$, $\boldsymbol{D}$ is actually a scalar linear function. By writing $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_c) = \sum_{i=1}^{c} \tau_i \boldsymbol{\tau}^i$, we can use the linearity of $\boldsymbol{d}$ and $\boldsymbol{D}$ to write the output of LEAP Net as

$$\begin{aligned}
\hat{\boldsymbol{y}} &= \boldsymbol{D}\left(\boldsymbol{E}(\boldsymbol{x}) + \boldsymbol{d}\left(\boldsymbol{e}(\boldsymbol{E}(\boldsymbol{x})) \odot \sum_{i=1}^{c} \tau_i \boldsymbol{\tau}^i\right)\right) \\
&= \boldsymbol{D}\left(\boldsymbol{E}(\boldsymbol{x}) + \sum_{i=1}^{c} \tau_i \boldsymbol{d}\left(\boldsymbol{e}(\boldsymbol{E}(\boldsymbol{x})) \odot \boldsymbol{\tau}^i\right)\right) \\
&= f_0(\boldsymbol{x}) + \sum_{i=1}^{c} \tau_i f_i(\boldsymbol{x})
\end{aligned}$$

where

$$\begin{aligned}
f_0(\boldsymbol{x}) &= \boldsymbol{D}\boldsymbol{E}(\boldsymbol{x}) \\
f_i(\boldsymbol{x}) &= \boldsymbol{D}\boldsymbol{d}\left(\boldsymbol{e}(\boldsymbol{E}(\boldsymbol{x})) \odot \boldsymbol{\tau}^i\right), \quad i = 1, \ldots, c.
\end{aligned}$$

As $NN(\boldsymbol{x}, \boldsymbol{\tau})$ makes perfect predictions for any data point $(\boldsymbol{x}, \boldsymbol{\tau}, \boldsymbol{y})$ coming from training domainswe have:

$$S(\boldsymbol{x}; \boldsymbol{\tau}) = N(\boldsymbol{x}, \boldsymbol{\tau})$$

which means that the following equalities hold

$$\begin{aligned}
F(\boldsymbol{x}) &= f_0(\boldsymbol{x}) \\
F(\boldsymbol{x}) + \varepsilon_i(\boldsymbol{x}) &= f_0(\boldsymbol{x}) + f_i(\boldsymbol{x}), \quad \forall i = 1, \ldots, c.
\end{aligned}$$

So we must have

$$\begin{aligned}
F(\boldsymbol{x}) &= f_0(\boldsymbol{x}) \\
\varepsilon_i(\boldsymbol{x}) &= f_i(\boldsymbol{x}), \quad \forall i = 1, \ldots, c.
\end{aligned}$$

for all $\boldsymbol{x}$. As in our case the distribution $D(\mathscr{X})$ of $\boldsymbol{x}$ does not depend on $\boldsymbol{\tau}$,the above equality holds for all $\boldsymbol{x}$.

So when we use the trained $NN(\boldsymbol{x}, \boldsymbol{\tau})$ to make predictions for different $\boldsymbol{\tau}$,

$$NN(\boldsymbol{x}, \boldsymbol{\tau}^{\mathscr{I}}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{c} \tau_i f_i(\boldsymbol{x}) = f_0(\boldsymbol{x}) + \sum_{i \in \mathscr{I}} f_i(\boldsymbol{x}) = F(\boldsymbol{x}) + \sum_{i \in \mathscr{I}} \varepsilon_i(\boldsymbol{x}) = S(\boldsymbol{x}, \boldsymbol{\tau}^{\mathscr{I}})$$

holds for any $\boldsymbol{\tau}^{\mathscr{I}}$, which concludes the proof. □

The above theorem shows that, under some conditions, LEAP nets are indeed capable of performing *super-generalization*: making good predictions on complex structural parameters $\boldsymbol{\tau}$ while it was only trained with unary cases $\boldsymbol{\tau}^i$. The fact that $\boldsymbol{d}$ and $\boldsymbol{D}$ are linear is essential for capturing the additivity of perturbations. For other types of perturbations such as the multiplicative case defined as follows

$$S(\boldsymbol{x}, \boldsymbol{\tau}^{\mathscr{I}}) = F(\boldsymbol{x}) \prod_{i \in \mathscr{I}} (1 + \varepsilon_i(x)),$$

LEAP nets can achieve similar results if $\boldsymbol{D}$ has, for example, an exponential-like behavior (*i.e.* it transforms additions into multiplications).

Despite its limitations, the fact that Theorem 2 works for any unknown functions $\varepsilon_i(\boldsymbol{x})$ makes the results very general. We can consider for example linear perturbations with $\varepsilon_i(\boldsymbol{x}) = \boldsymbol{W}_i \boldsymbol{x}$, constant additive perturbations with $\varepsilon_i(\boldsymbol{x}) = \boldsymbol{\alpha}_i$ or heteroskedastic perturbations with $\varepsilon_i(\boldsymbol{x}) = \boldsymbol{\alpha}_i G(\boldsymbol{z})$, etc. And all these types of perturbation are special cases of the above theorem.

Theorem 2 gives a strong theoretical guarantee, but one may wonder if the condition of making *perfect* predictions can be too restrictive. So we also investigated the slightly more general case of *imperfect* predictions, when the LEAP net is trained only on unary changes.

To do this, we first need to introduce a notion of distance between functions. As we used Mean Square Error (MSE) in our work for the regression problem, the distance we use will be defined in the same flavor. Let $\mathscr{X} \subseteq \mathbb{R}^p$ be the support of $\boldsymbol{x}$, $\mathscr{Y} \subseteq \mathbb{R}^l$ be the support of $\boldsymbol{y}$. Let μ be a probability measure on $\mathscr{X}$, and f, g two functions from $\mathscr{X}$ to $\mathscr{Y}$.

We define the distance $d_\mu(f,g)$ between f and g to be :

$$d_\mu^2(f,g) = \int_{\mathscr{X}} \|f(\boldsymbol{x}) - g(\boldsymbol{x})\|^2 d\mu(\boldsymbol{x}) \tag{6.11}$$

Here $\|\cdot\|$ represents the ℓ_2-norm on $\mathbb{R}^l$. Notice that this distance depends on the probability measure μ. If we put $\mu = D(\mathscr{X})$ (the ground truth distribution of $\boldsymbol{x}$) and write $\boldsymbol{y} = g(\boldsymbol{x})$, the right hand side becomes nothing but the generalization error [4] of f. If we instead select $\mu = \frac{1}{m}\sum_{i=1}^m \delta_{\boldsymbol{x}_i}$ (the empirical distribution), the right hand side becomes the MSE test error (or training error). We are now ready to formulate our theorem on the imperfect prediction case.

Theorem 3. *Let $S(\boldsymbol{x},\boldsymbol{\tau})$ be a system satisfying Equations 6.9 and 6.10. Let $NN(\boldsymbol{x},\boldsymbol{\tau})$ be a LEAP net with linear submodules $\boldsymbol{d}$ and $\boldsymbol{D}$ such that*

$$\begin{aligned} d_\mu\left(NN(\cdot,\boldsymbol{\tau}^{\emptyset}), S(\cdot,\boldsymbol{\tau}^{\emptyset})\right) &\leq d_0 \\ d_\mu\left(NN(\cdot,\boldsymbol{\tau}^{i}), S(\cdot,\boldsymbol{\tau}^{i})\right) &\leq d_i, \quad i = 1,\ldots,c. \end{aligned} \tag{6.12}$$

for some constant $d_0, d_1, \ldots, d_c \in \mathbb{R}$. Then, for any $\mathscr{I} \subset \{1,\ldots,c\}$, we have

$$d_\mu\left(NN(\cdot,\boldsymbol{\tau}^{\mathscr{I}}), S(\cdot,\boldsymbol{\tau}^{\mathscr{I}})\right) \leq (|\mathscr{I}|+1)d_0 + \sum_{i\in\mathscr{I}} d_i. \tag{6.13}$$

Proof. According to (6.9) and (6.10) in Theorem 2, we can write $S(\cdot,\boldsymbol{\tau}^{\emptyset}) = F$ and $S(\cdot,\boldsymbol{\tau}^{i}) = F + \varepsilon_i$. And since $\boldsymbol{d}$ and $\boldsymbol{D}$ are linear, we can write $NN(\cdot,\boldsymbol{\tau}^{\emptyset}) = f_0$ and $NN(\cdot,\boldsymbol{\tau}^{i}) = f_0 + f_i$ according to the same argument in the proof of Theorem 2. Then we can rewrite (6.12) as

$$\begin{aligned} d_\mu\left(f_0, F\right) &\leq d_0 \\ d_\mu\left(f_0 + f_i, F + \varepsilon_i\right) &\leq d_i, \quad i = 1,\ldots,c. \end{aligned}$$

4. We remind that in our problem setting, the system $S(\cdot,\boldsymbol{\tau})$ we want to identify is deterministic (solutions of differential equations), the labels $\boldsymbol{y}$ are thus deterministic given $\boldsymbol{x}$ and $\boldsymbol{\tau}$. So we don't need to consider the joint distribution on the pair $(\boldsymbol{x},\boldsymbol{y})$.

Because the distance d_μ defined above satisfies triangle inequality (and is translation invariant), we have

$$\begin{aligned}
d_\mu\left(NN(\cdot,\boldsymbol{\tau}^{\mathscr{I}}),S(\cdot,\boldsymbol{\tau}^{\mathscr{I}})\right) &= d_\mu\left(f_0+\sum_{i\in\mathscr{I}} f_i, F+\sum_{i\in\mathscr{I}}\varepsilon_i\right)\\
&= d_\mu\left(f_0+\sum_{i\in\mathscr{I}}[(f_0+f_i)-f_0], F+\sum_{i\in\mathscr{I}}[(F+\varepsilon_i)-F]\right)\\
&\leq d_\mu(f_0,F)+\sum_{i\in\mathscr{I}} d_\mu(f_0+f_i,F+\varepsilon_i)+\sum_{i\in\mathscr{I}} d_\mu(f_0,F)\\
&\leq d_0+\sum_{i\in\mathscr{I}} d_i+\sum_{i\in\mathscr{I}} d_0\\
&= (|\mathscr{I}|+1)d_0+\sum_{i\in\mathscr{I}} d_i,
\end{aligned}$$

which concludes the proof. □

Theorem 3 shows that as long as a LEAP net approximates well the ground truth functions 6.9 by fitting data from a given distribution of $(\boldsymbol{x},\boldsymbol{\tau})$, its error on any points where $\boldsymbol{\tau}$ is altered and results in combination of the vector observed in the training set can be bounded too. This can indeed be considered as a super-generalization property. Notice that the weight of d_0 is larger than that of the others, which suggests we should make more efforts on improving the accuracy of the predictions on the reference topology case $\boldsymbol{\tau}=\boldsymbol{\tau}^{\emptyset}$. This was the case in all our experiments where more data came from this distribution.

Obviously, Theorem 2 is a special case of Theorem 3 with $d_0=d_1=\cdots=d_c=0$.

At last, we emphasize that the distance d_μ is a very flexible notion as it can be defined as the generalization error or more importantly in practice, as the test error.

6.3.5 . Connection with Transfer Learning and Meta-learning

In this work of LEAP we have cast our problem as a machine learning problem from iid data, lumping distributional changes in our objective function, which re-balances the importance of various values of $\boldsymbol{\tau}$. Another angle would be to treat our problem as a true change in distribution. The issue of having to deal with changes in distributions has been

extensively studied in the framework of transfer learning (TL) (Goodfellow et al., 2016; Pan and Yang, 2010)[5]. To make the analogy with our problem setting, we can think of our reference case $\boldsymbol{\tau}^{\emptyset}$ as a *source domain*, and other $\boldsymbol{\tau}$ values as *target domains.*[6] Future work includes using LEAP nets to model changes in distributions in which $\boldsymbol{\tau}$ is a latent variable rather than an actionable variable.

Among many cases in the taxonomy of TL (Pan and Yang, 2010) we discuss here two cases closely related to LEAP nets: Domain Adaptation (DA) and Multi-Task learning (MT). To make the connection, it is useful to consider two possible factorizations of the joint distribution of inputs and outputs:

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{x})P(\boldsymbol{y}|\boldsymbol{x}) = P(\boldsymbol{x}|\boldsymbol{y})P(\boldsymbol{y}) \tag{6.14}$$

DA and TL make different assumptions on what changes in $P(\boldsymbol{x}, \boldsymbol{y})$ between source and target domains: DA assumes that $P(\boldsymbol{x})$ changes, but $P(\boldsymbol{y}|\boldsymbol{x})$ remains unchanged. MT assumes that $P(\boldsymbol{y})$ changes, but $P(\boldsymbol{x}|\boldsymbol{y})$ remains unchanged. Behind that are some implicit causal assumptions about the data generative process, which we will discuss in the following. Obviously there are cases in which variables in $\boldsymbol{x}$ and $\boldsymbol{y}$ are inter-related in a complex manner and this simple factorization does not make sense, but here we limit ourselves to a discussion of DA and MT.

For our power network application setting, DA has a more natural interpretation since $P(\boldsymbol{x}, \boldsymbol{\tau})$ changes, but $P(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\tau})$ does not (in fact, it is a deterministic function). But both groupings of variables are possible and our LEAP net architecture does not preclude of either.

As illustrated in Figure 6.11, we argue that, in applications to both DA and TL problems, LEAP nets could present distinct advantages over other architectures because of the flexibility of making leaps induced

5. We recall that due to the generality of our definition of meta-learning, transfer learning is a special case of *second-order meta-learning* (i.e. S2), with only one single task in the meta-training set

6. Alternatively, the reference case and unary changes can be lumped in the source domain, then combinations of changes would make the target domain. In this context, the LEAP net architecture, as exemplified in our experimental section, can be interpreted as an architecture capable of "zero shot learning", i.e. learning combinations of perturbations from zero examples. Structural variable $\boldsymbol{\tau}$ should be interpreted as a latent variable (rather than an actionable variable) and may take continuous values.

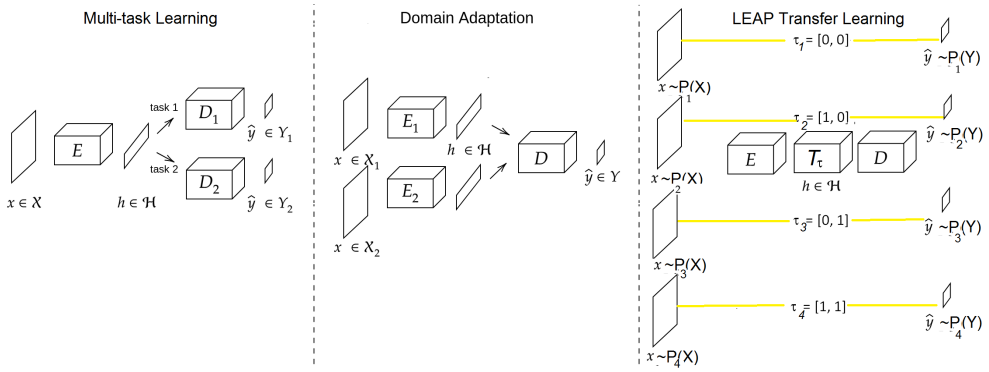


Figure 6.11 **Different types of transfer learning**. From left to right: Multi-Task learning or MT (the encoder $\boldsymbol{E}$ is common to both tasks, which have the same input domain, but the decoders $\boldsymbol{D}_1$ and $\boldsymbol{D}_2$ are task-specific); Domain Adaptation or DA (each input domain has a separate encoder $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$ creating a common embedding $\boldsymbol{h} \in \mathscr{H}$ then processed by $\boldsymbol{D}$, i.e. we have different domains but the same task); LEAP transfer learning (our proposed setting: a combination of MT and DA in which changes in domain are encoded as a LEAP in $\mathscr{H}$ permitting to combine combinatorially unary changes).

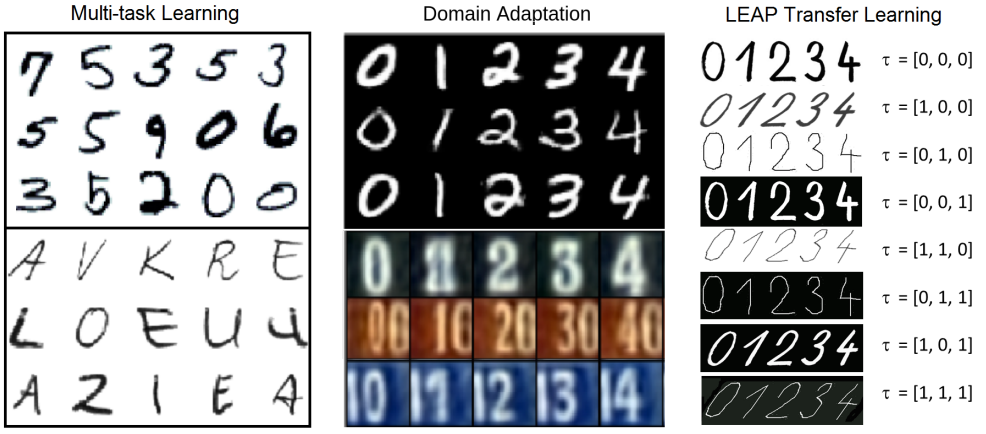


Figure 6.12 **Examples of transfer learning**. This illustrates with some handwriting recognition examples the different types of transfer learning of Figure 6.11. For Multi-task Learning and Domain Adaptation, we have a simple division between source domain (top) and target domain (bottom). For LEAP transfer learning, we have a reference source domain for which $\tau = [0,0,0]$ and multiple target domains, some corresponding to unary changes ($\tau = [1,0,0] \rightarrow$slanted, $\tau = [0,1,0] \rightarrow$skeletonized, $\tau = [0,0,1] \rightarrow$inverted) and some to combinations of changes ($\tau = [1,1,0]$,$\tau = [0,1,1]$,$\tau = [1,0,1]$,$\tau = [1,1,1]$). The avantage of this setting is that we can train on unary changes and obtain super-generalization on combinations of changes never seen during learning.

by $\boldsymbol{\tau}$ in latent space encoding combinations of unary perturbations (Figure 6.12).

Please note that some of the values of $\boldsymbol{\tau}$ in our examples of digit distortions could be continuous (*e.g.* continuous change in skew).

In many ways the LEAP framework generalizes both DA and TL, as further illustrated in Table 6.4. The TL taxonomy (Pan and Yang, 2010) also separate various types of TL according to the availability of labeled examples in the source and/or target domains. The LEAP net architecture, as exemplified in our experimental section, lends itself to "zero shot learning", i.e. learning from zero examples in the target domain. This is what we have called "super-generalization" in the rest of the work.

Finally this analysis allows us to position this work in the context of meta-learning taxonomy. The $\boldsymbol{\tau}$ hyperparameter can be interpreted as a meta-feature hence the search space is S1 (we don't need all concrete examples since we can generate them from $\boldsymbol{\tau}$). The LEAP nets, once trained, are capable of super-generalization, i.e. to make predictions for new $\boldsymbol{\tau}$ without re-training. Hence the meta-learning procedure directly returns an α-level algorithm, which is an A0 action. Finally the objective of training a surrogate model is to improve speed of simulation without sacrificing accuracy. Hence the reward is $RC|RP,RE$.

6.4 . Theoretical Analysis of Zero-order Meta-learning

In this section, we make an attempt to prove some theoretical results for meta-learning.

We consider a simplified meta-learning setting where only a matrix of performances is available, i.e. we are in the *zero-order meta-learning setting*. The corresponding SAR tags of this section are $S0, A1, RC|RP,RE$, i.e. only performances in the DA matrix are considered (no detailed information or meta-features of the tasks and the algorithms) ($S0$) and the action in each iteration is to choose a β-level algorithm ($A1$). Then one wishes to improve computational aspect by maintaining the performance/accuracy and the number of examples.

Concretely, we have access to a meta-dataset of past performances (we use the notation from Chapter 3)

$$\mathfrak{D}_{tr} = \{(T_j, \beta_j, R_j)\}_{j \in J}$$

with $J = \{1, ..., N\}$ where the tuple (T_j, β_j) traverses a finite set $\mathscr{T} \times \mathscr{B}$ for $T = \{T_1, \ldots, T_n\}$ and $\mathscr{B} = \{\beta_1, \ldots, \beta_p\}$. Furthermore, we only consider $R_j \in \{0, 1\}$ with $R_j = 0$ indicating that we applied a 'good' learning algorithm on the task/dataset and $R_j = 1$ otherwise. All above hypotheses reduce our situation to the case where we only know a performance matrix

$$R \in \{0, 1\}^{n \times p}$$

for carrying out meta-learning. Recall from Section 2.2.3 that we call this matrix the *DA matrix* since dataset-algorithm pairs are considered. This is the setting adopted by (Sun-Hosoya, 2019).

Now the question is, when a new task T_{n+1} (from a meta-test set $\mathfrak{D}_{te}$) arrives, which algorithm in $\mathscr{B} = \{\beta_1, \ldots, \beta_p\}$ should one recommend such that it is probably a 'good' algorithm on this new task? We consider an agent that applies each algorithm β_j on T_{n+1} one by one following certain order. This order is denoted by $t_j \in \{1, \ldots, p\}$ for each step $j = 1, 2, ..., p$. A visualization of the setting of the problem is shown in Figure 6.13.

	β_1	β_2	$\cdots$	β_p
T_1	R_{11}	R_{12}	$\cdots$	R_{1p}
T_2	R_{21}	R_{22}	$\cdots$	R_{2p}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
T_n	R_{n1}	R_{n2}	$\cdots$	R_{np}
T_{n+1}	$j_2 = 1$	$j_1 = 2$		

R

Figure 6.13 **DA matrix for meta-learning with zero-order information.** R_{ij} is the performance obtained by applying the algorithm R_i on the dataset/task T_j. The problem to recommend one algorithm for a new task T_{n+1}.

6.4.1 . Notations and Problem Setting

The DA Matrix

We consider the setting in which a DA matrix is available for meta-learning, that is a matrix of performances of algorithms on datasets/tasks. To simplify the analysis, we assume that scores are **binary**: 0 means algorithm failure and 1 algorithm success. We further assume that tasks/datasets (lines of the DA matrix) are drawn *i.i.d.* according to an unknown but fixed meta-distribution. Performances of algorithms on tasks/datasets can be thought of as random variables R_j, $j = 1 \dots n$, where n is the total number of algorithms considered. We call $\mathbf{R}$ the random vector $[R_1, R_2, \dots, R_n]$.

The goal of meta-learning is to learn from m samples of $\mathbf{R}$ constituting a training DA matrix, to devise a **meta-predict strategy**. After learning, this strategy is applied to find a successful algorithm as fast as possible, given a new task/dataset not seen before, *i.e.* . by querying the performance of as few algorithms as possible.

Here we assume that we have an infinite number of training examples, such that the joint distribution $P(R_1, R_2, \dots, R_n)$ is known perfectly. Hence the **meta-training** procedures considered search for an optimal order of algorithms, knowing the meta-distribution perfectly. We ask ourselves the following questions: (i) Does the perfect knowledge of $P(R_1, R_2, \dots, R_n)$ allow us to outperform random search. (ii) Under what conditions (if any).

Criterion of evaluation

To evaluate the performance of any given meta-predict strategy, we consider as metric the area under the (meta-)learning curve. A learning curve $lc(i)$, $i = 1 \dots n$, is defined as the performance of the best algorithm queried so far, as a function of the number of algorithms queried. That is:

$$lc(i) = \max_{t_j, j=1 \dots i} \{R_{t_1}, R_{t_2}, \dots, R_{t_i}\} , \tag{6.15}$$

where we denote by R_{t_i} the score of the i^{th} algorithm queried. Next, we adopt a probabilistic notion of learning curve, considering a meta-test example (new task or dataset) as a random variable, *i.e.* the ex-

pectation of $lc(t_i)$ over possible meta-test examples (a form of meta-generalisation):

$$LC(i) = \mathbb{E}\left[\max_{t_j, j=1\ldots i} \{R_{t_1}, R_{t_2}, \ldots, R_{t_i}\}\right] . \tag{6.16}$$

Given that scores are either 0 or 1, the expected value of the maximum score seen so far is the probability that at least one algorithm was successful (score 1). This is also one minus the probability that all algorithms seen so far failed:

$$LC(i) = 1 - P(R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_i} = 0) . \tag{6.17}$$

To evaluate the performance of meta-predict strategies having a stochastic component, we define

$$LC(i) = \mathbb{E}\left[1 - P(R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_i} = 0)\right] \tag{6.18}$$

where the expectation runs over possible algorithm orderings. For strategies with a fixed pre-determined order of algorithms to be queried, taking this second expectation is not necessary. It is necessary for the Random strategy and when ties are broken at random.

Finally, we define the area under the learning curve, which we wish to maximize it over meta-predict strategies:

$$ALC(n) = \sum_{i=1}^{n} LC(i) . \tag{6.19}$$

Meta-predict strategies

We consider four meta-predict strategies:

— **Random:** Query algorithms in uniformly random order.

— **Mean:** Query algorithms in order of their mean score value $\mathbb{E}[R_j] = P(R_j = 1)$.

— **Greedy:** Query first the algorithm with largest $P(R_{t_1} = 1)$. Then, query iteratively the next algorithm having the largest $P(R_{t_i} = 1 | R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_{i-1}} = 0)$, until we find one successful algorithm; then the order of the remaining algorithms does not matter.

— **Optimal:** Query algorithms in the optimal order, maximizing $ALC(n)$.

For the Mean, Greedy, and Optimal strategies, we assume that ties are broken at random. When the number of training examples is finite (as what we have in the Empirical Results section), all above probabilities can be empirically estimated by the corresponding average.

Meta-distributions

We consider four types of meta-distributions:

— **NFL:** No Free Lunch distribution: $P(R_1, R_2, \ldots, R_n) = P(R_1)P(R_2)\ldots P(R_n)$ and $P(R_j) = 0.5, \forall j = 1 : n$.

— **Indep:** $P(R_1, R_2, \ldots, R_n) = P(R_1)P(R_2)\ldots P(R_n)$ but for some $j\, P(R_j) \neq 0.5$. To make it more comparable to NFL, we assume that $(1/n)\sum_{j=1}^{n} P(R_j) = 0.5$.

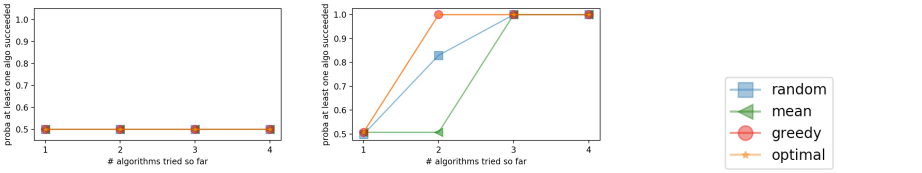
— **Dep:** $P(R_1, R_2, \ldots, R_n) \neq P(R_1)P(R_2)\ldots P(R_n)$. We keep assuming $(1/n)\sum_{j=1}^{n} P(R_j) = 0.5$.

— **DepU: Dep** with uniform marginals $P(R_j) = 0.5, \forall j$.

6.4.2 . Theoretical Results

In what follows, we prove the following propositions:

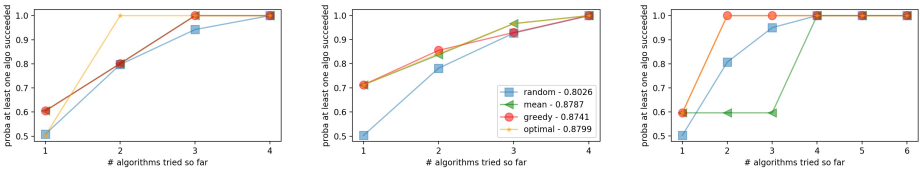
1. For the **NFL** distribution, the **Random** meta-predict strategy is as good as anything else.
2. For **Indep**, the **Mean** strategy is optimum.
3. For the **Dep** distribution, we might expect that the performance order should be **Random** $\leq$ **Mean** $\leq$ **Greedy** $\leq$ **Optimal**, however a variety of cases can arise:
   (a) **"Worst case"**: All strategies perform at chance level.
   (b) **"Best case"**: Greedy is optimal.
   (c) **Greedy** makes best "local" decisions, *i.e.* increasing most the learning curve at any given point.
   (d) **Greedy** can be worse than **Optimal**
   (e) **Greedy** can be worse than **Random**
   (f) **Greedy** can be worse than **Mean**



(a) 3.a: Worst case scenario

(b) 3.b: Best case scenario

(c) Legend



(d) 3.d: Greedy worse than Optimal

(e) 3.f: Greedy worse than Mean.

(f) 3.g: Mean worse than Random

Figure 6.14 **Learning curves on constructed examples.** (a) **3.a: "Worst-case" scenario:** All algorithms are identical. Random search is as good as anything else. (b) **3.b: "Best-case" scenario:** Two algorithms are exactly complementary (one succeeds when the other one fails). All other algorithms are independent of the two first ones and of one another. Greedy is optimal. (d) **3.d: Greedy worse than Optimal:** Both mean and greedy do not choose optimally the first point: it is the best performing algorithm by itself, but does not belong to the best performing pair. (f) **3.f: Greedy worse than Mean:** Mean provides by coincidence the optimal order, which Greedy does not select. We use $\varepsilon = 0.1$ (see text). ALCs are shown in legend. (g) **3.g: Mean worse than Random:** Redundant versions of the best performing algorithm are included. Mean ranks them all first rather than selecting complementary algorithm.

(g) **Mean** can be worse than **Random**

4. For the **DepU** distribution, the **Mean** meta-predict strategy performs no better than **Random**. We define a notion of meta-learning complexity C as the cardinal of the minimal clique of complementary columns (*i.e.* columns having at least one successful algorithm in each line). There exists a meta-predict strategy such that $LC(C) = 1$, *i.e.* the learning curve asymptote is reached in C steps.

6.4.3 . Proofs

We simplify notations using: $P(R_{t_i} = 1) = p_i = 1 - q_i$. For **NFL** and **Indep** meta-distributions, learning curves are:

$$\begin{aligned} LC(i) &= 1 - P(R_{t_1} = 0)P(R_{t_2} = 0)\dots P(R_{t_i} = 0) \\ &= 1 - q_1 q_2 \dots q_i \end{aligned} \tag{6.20}$$

1. For the NFL distribution, the Random meta-predict strategy is as good as anything else.

For the **NFL** meta-distributions, $p_i = 1 - q_i = 0.5$. Thus,

$$LC(i) = 1 - (0.5)^i \,, \tag{6.21}$$

for all strategies. Thus **Random** is as good as anything else.

2. For Indep, Mean is optimum.

For the **Indep** distribution, the **Mean** strategy provides a ranking such that $q_1 \leq q_2 \leq \dots \leq q_i$. Therefore the product $q_1 q_2 \dots q_i$ for the **Mean** strategy will be smaller (or equal) to that obtained for any other order of algorithms.

$$\begin{aligned} LC(i) &= 1 - q_1 q_2 \dots q_i \\ &\leq 1 - (0.5)^i \end{aligned} \tag{6.22}$$

3.a. Worse case scenario: All strategies perform at chance level

This case arises simply when all algorithms are identical: $R_1 = R_2 = \dots = R_n$. In that case, by our previous assumption that on average algorithms perform at chance level, we have $P(R_j = 1) = 0.5$. For individual tasks, the (unique) algorithm we have will either be successful or fail, hence $lc(i) = 0$ of all i or $lc(i) = 1$ of all i. Thus $LC(i) = 0.5$ for all i, regardless of the strategy chosen.

3.b. Best case scenario: Greedy is optimal

This case arises, for example, when we have only 2 types of algorithms: an algorithm with scores R_1, and another one with exactly

complementary scores: $R_2 = 1 - R_1$. All n algorithms are either of the first or the second type. We recall that by hypothesis, on average, algorithms perform at chance level. When we query a first algorithm, regardless of the strategy, predictions are therefore at chance level $LC(1) = 0.5$. But, as soon as we query a second algorithm with the greedy strategy, we get $LC(i > 1) = 1$ because $P(R_2 = 1|R_1 = 0) = 1$ and $P(R_1 = 1|R_2 = 0) = 1$, hence the **Greedy** strategy will pick up one of the versions of the complementary algorithm. The **Optimal** algorithm cannot beat it because $LC(1) = 0.5$ regardless of strategy.

3.c. Greedy makes the best local decision

This can be straightforwardly proven by the definition of **Greedy**. The best local decision is the decision that increases most the learning curve at any given point. By definition,

$$\begin{aligned} LC(i) &= 1 - P(R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_i} = 0) = 1 - \mathscr{P} \\ &= 1 - P(R_{t_i} = 0 | R_{t_1} = 0, \ldots, R_{t_{i-1}} = 0) \\ &\qquad \cdot P(R_{t_1} = 0, \ldots, R_{t_{i-1}} = 0)) \ . \end{aligned} \tag{6.23}$$

Hence the variation of learning curve is:

$$\begin{aligned} \Delta LC(i) &= LC(i+1) - LC(i) \\ &= \mathscr{P} \cdot (1 - P(R_{t_{i+1}} = 0 | R_{t_1} = 0, \ldots, R_{t_i} = 0) \\ &= \mathscr{P} \cdot P(R_{t_i} = 1 | R_{t_1} = 0, \ldots, R_{t_i} = 0) \end{aligned} \tag{6.24}$$

By definition of the greedy strategy, unless the learning curve has already reached its maximum value, Greedy chooses at step i the algorithm with the largest $P(R_{t_i} = 1|R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_i} = 0)$. Hence greedy makes the choice that increases most the learning curve, given past decisions, and ignoring what will happen in the future.

3.d. Greedy can be worse than Optimal

In this example we exploit the fact that the first step of the **Greedy** strategy is to choose the algorithm with best mean score (the same

as the **Mean** strategy). But this algorithm is not necessarily the most informative about what second algorithm should be chosen.

Assume that we have four types of algorithms: R_1, R_2, R_3, and $R_4 = 1 - R_3$, with $P(R_1 = 1) = 0.5 + \varepsilon$ and $P(R_2 = 1) = 0.5 - \varepsilon$, $0 < \varepsilon \ll 1$, $P(R_3 = 1) = P(R_4 = 1) = 0.5$, $R_1 \perp\!\!\!\perp R_2 \perp\!\!\!\perp (R_3|R_4)$, $R_3 \not\perp\!\!\!\perp R_4$.

Greedy will choose algorithm 1 first, then R_3 or R_4 without distinction (assume it chooses R_3), then finally it will choose R_4 (as perfectly complementing R_3) and R_2 last. Therefore the learning curve of **Greedy** will be:

$$LC(1) \simeq 0.5, \;\; LC(2) \simeq 1 - (0.5)^2, \;\; LC(i \geq 3) = 1 \; .$$

In contrast the optimal strategy is to choose R_3 or R_4 first, without distinction. Assume it chooses R_3 first, then it will choose R_4 and reaches perfect prediction in only two steps. Therefore the learning curve of **Optimal** will be:

$$LC(1) \simeq 0.5, \;\; LC(i \geq 2) = 1 \; ,$$

and therefore **Optimal** is better than **Greedy**.

3.e. Greedy can be worse than Random

True since Greedy can be worse that Optimal and Random can reach Optimal by chance. However, it is more interesting to find out whether Greedy can perform well in "most cases" than Random does on average, since Greedy is a deterministic algorithm, while Random has a lot of variance. See Sections Empirical Results and Computational Considerations.

3.f. Greedy can be worse than Mean

In this example, we exploit the fact that "by chance", **Mean**, which is ordering its algorithms with the marginal probabilities, would order them in an optimal order, while **Greedy** would choose a sub-optimal order. This construction is possible with at least 4 algorithms that we call A, B, C, and D.

We recall that the construction is possible with at least 4 algorithms that we call A, B, C, and D. Without loss of generality, we assume that $P(A=0) < P(B=0) < P(C=0) < P(D=0)$ (no ties). In this example we chose that:

— $P(A=0) = 0.5 - 2\varepsilon$, $P(B=0) = 0.5 - \varepsilon$, $P(C=0) = 0.5 + \varepsilon$, $P(D=0) = 0.5 + 2\varepsilon$, where $0 < \varepsilon \ll 1$.

— $P(A=0, B=0, C=0) = 0$. Note that this also means that $P(A=0, B=0, C=0, D=0) = 0$.

The example leads **Mean** to choose the optimal order A, B, C, D that reaches the asymptote of the learning curve in 3 steps, whereas **Greedy** does not: **Greedy** chooses A, D, C, B. This can happen if we have the following conditional probabilities:

Step	**Mean**	**Greedy**
1	$P(A=0) = 0.5 - 2\varepsilon$	$P(A=0) = 0.5 - 2\varepsilon$
2	$P(B=0 \mid A=0) = 0.5 + 2\varepsilon$	$P(D=0 \mid A=0) = 0.5$
		$P(C=0 \mid A=0) = 0.5 + 2\varepsilon$
3	$P(C=0 \mid A=0, B=0) = 0$	$P(C=0 \mid A=0, D=0) = 0.5$
		$P(B=0 \mid A=0, D=0) = 0.5$

With these values the learning curve cross each other. We can compute the learning curves values and the ALC:

$$LC(i) = 1 - P(R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_i} = 0) \ .$$

$$ALC = \sum_{i=1}^{n} LC(i)$$

	Mean	**Greedy**
$1 - LC(1)$	$0.5 - 2\varepsilon$	$0.5 - 2\varepsilon$
$1 - LC(2)$	$(0.5 - 2\varepsilon)(0.5 + 2\varepsilon)$	$(0.5 - 2\varepsilon) \cdot 0.5$
$1 - LC(3)$	0	$(0.5 - 2\varepsilon) \cdot 0.5 \cdot 0.5$
$1 - LC(4)$	0	0

Let us verify that ALC(**Mean**) $>$ ALC(**Greedy**). ALC(**Mean**) $>$ ALC(**Greedy**), if and only if $LC(2) + LC(3)$ is larger for **Mean** than for **Greedy** since $LC(1)$ and $LC(4)$ are identical. Equivalently, $2 - LC(2) - LC(3)$ is smaller

for **Mean** than for **Greedy**. We have

$$\begin{aligned} & ALC(\textbf{Mean}) - ALC(\textbf{Greedy}) \\ & = (0.5-2\varepsilon)\cdot 0.5^2 + (0.5-2\varepsilon)\cdot 0.5 - (0.5-2\varepsilon)(0.5+2\varepsilon) \\ & = 1/8 - \varepsilon/2 + 1/4 - \varepsilon - 1/4 + 4\varepsilon^2 \\ & = 1/8 - \frac{3}{2}\varepsilon + 4\varepsilon^2 \end{aligned} \tag{6.25}$$

which is positive for small enough ε. For *e.g.* $\varepsilon = 0.1$, **Mean** has a better ALC than **Greedy** (we get $0.015 > 0$).

3.g. Mean can be worse than Random

This case arises in a similar example as example 3.b. We still have $R_2 = 1 - R_1$. But we assume that there is a small difference in the average score of two types of algorithms considered: $P(R_1) = 0.5 + \varepsilon$ and $P(R_2) = 0.5 - \varepsilon$, with $0 < \varepsilon \ll 1$. In this case, the **Mean** strategy will rank first all the algorithms of type 1. Hence for the **Mean** strategy:

$$LC(i \leq n/2) \simeq 0.5,\ LC(i > n/2) = 1\ . \tag{6.26}$$

For the random strategy, we also have $LC(i > n/2) = 1$ because half of the algorithms are of a complementary type, so even in case of extreme bad luck where we draw first all the algorithms that fail on a particular task, at $n/2+1$ we get one that succeeds (its complement). For the **Random** strategy, between at each step, we increase our probability of getting a good algorithm, yielding the learning curve:

$$LC(i \leq n/2) \simeq 1 - 0.5^i,\ LC(i > n/2) = 1\ . \tag{6.27}$$

4. For the DepU distribution, the Optimal algorithm attains $LC = 1$ in C steps.

First, note that, for the DepU distribution, since all marginal distributions are identical (average performance of algorithms identical), the Mean strategy performs like the Random strategy (since ties are broken at random).

A set of algorithms $\{R_j\}, j = 1 \cdots p$ will be called **complementary** if and only if $P(R_1 = 0, R_2 = 0, \cdots, R_p = 0) = 0$. Hence, for each task, there exists at least one successful algorithm in that set. Further, we call a **clique** a minimal set of complementary algorithms, *i.e.* such that removing any of its members breaks complementarity. Finally, we define a notion of **complexity of a meta-learning problem** C as the **cardinal of the smallest clique** (when there is one), and $C = n$ otherwise.

For a meta-learning problem of complexity C, there exists an meta-predict strategy such that $LC(C) = 1$. Indeed, it suffices to rank first the algorithms of the smallest clique. Note however that neither the Greedy not the Optimal strategies attain necessarily $LC(C) = 1$. Nonetheless, we will see in the experimental section (Section Empirical Resutls) that the smaller C, the larger the ALC of Greedy and Optimal.

6.4.4 . Empirical Results

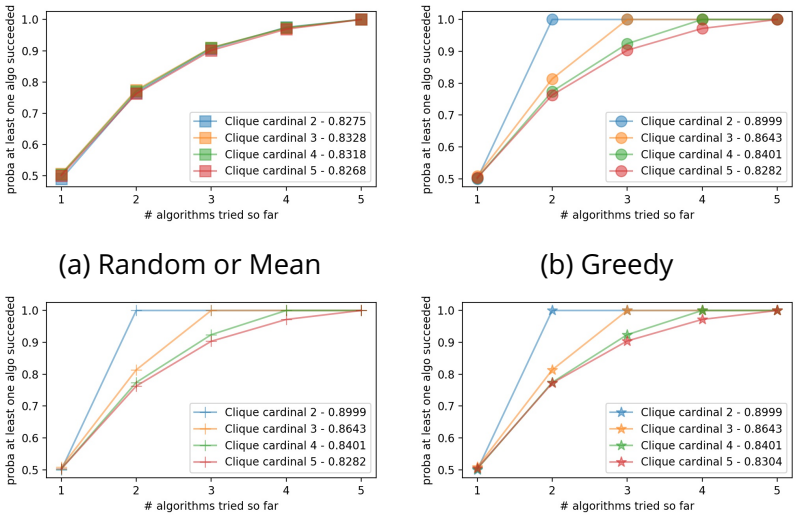
In this section, we compare the four meta-predict strategies considered on various meta-distributions. The theoretical results offer no guarantee of optimality of the **Greedy** method. But we offer empirical evidence of its effectiveness.

First we report results on synthetic data constructed such that the meta-distribution includes a single clique, and we vary the complexity C (size of the clique). Specifically, a DA matrix is constructed as follows: All values are initialized with -1 (missing); for C of its columns, a 1 is randomly placed in each line; the remaining -1 values are replaced by 0 or 1 randomly, such that the average values of each column is 0.5. In these experiments we use $n = 5$ and $m = 10000$ both for training and testing.

Figure 6.15 shows the learning curves for the various meta-predict strategies when C varies. We see that ALC performance does not change with C for the Random, Mean, and vanilla Greedy algorithms. It does improve for smaller C for the optimal strategy. We introduce a variant of the Greedy strategy called Greedy+ in which, at meta-training time, all algorithms are tried for the first position, then greedy search is performed. This algorithm is more computationally costly at meta-training time, but it results in a single algorithm ordering, hence is not

more costly at meta-predict time. Greedy+ performs nearly as well as Optimal.

Figure 6.16 shows the relationship between the ALC (Area under Learning Curve) and complexity C.



(a) Random or Mean

(b) Greedy

(c) Greedy+

(d) Optimal

Figure 6.15 **Learning curves for toy data, for varying complexity (clique cardinal).** All marginals (ave. algo. perf.) are identical to 0.5. Hence Mean has not advantage over Random. More subtly, neither does Greedy on average. Greedy+ however selects first the algorithms of the clique and performs as well as Optimal. ALC shown next to the clique cardinal in legend.

To illustrate the behavior of the meta-predict strategies considered, we also report results on benchmark meta-datasets used in (Sun-Hosoya et al., 2018b). These meta-datasets are Statlog (Australian Credit Approval) Data Set (21 tasks 24 algorithms), AutoML challenge dataset (Guyon et al., 2015, 2018) (30 tasks 17 algorithms), a subset of OpenML (Van Rijn et al., 2013; Vanschoren et al., 2014) (76 datasets 292 algorithms) and an artificially generated dataset by (Sun-Hosoya et al., 2018b) (50 tasks 20 algorithms). Since our setting considers only binary algorithm scores (failure or success), we binarized the meta-datasets (using the median value as threshold). We omit the Optimal strategy due to its prohibitive computational requirement, with a time complexity of $O(n!)$ (where n is the number of algorithms). Meta-predict

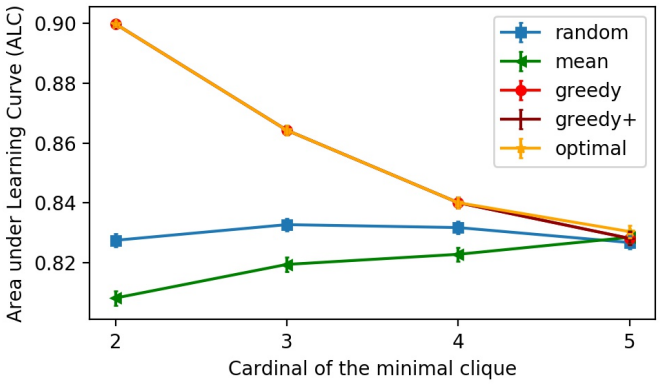


Figure 6.16 **ALC vs Clique cardinal C.** For Random, Mean, and Greeedy, the ALC does not vary (within the experimental error bar $\simeq VAL$. For Greedy+ and Optimal, the ALC decreases with C.

performances are evaluated using the leave-one-dataset out estimator, as in the original paper.

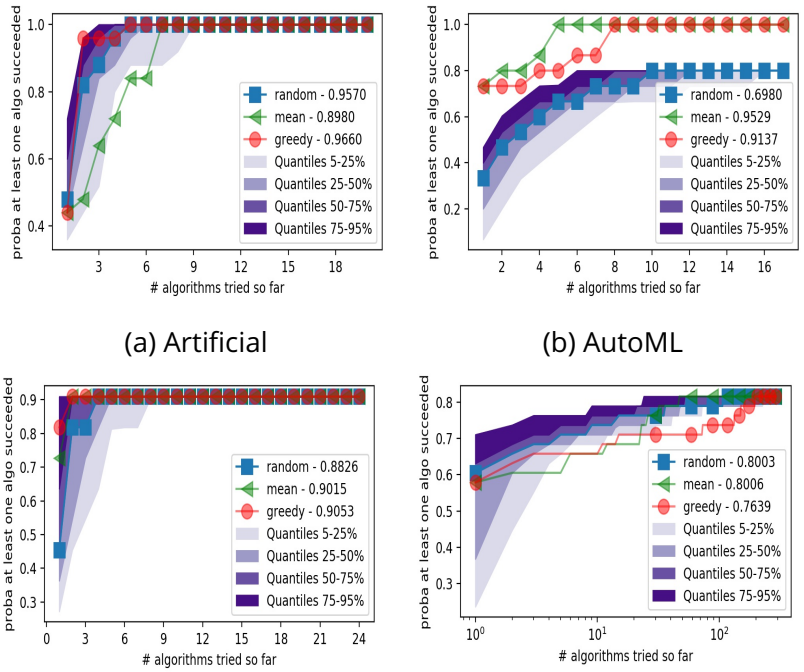
The learning curves, shown in Figure 6.17, are qualitatively similar to those of the original paper (Sun-Hosoya et al., 2018b). Remarkably, for the first dataset (Figure 6.17a), the performance of Mean is worse than others both in the original and the binarized data. Notably, the Greedy algorithm generally performs best (or closely as well as the best), and always better than the Random strategy does on average.

6.4.5 . Computational Considerations

Although this paper focuses on asymptotic analyses (infinite sample limit), for all practical purposes, we must evaluate the conditional probabilities from data, *i.e.* a finite set of m training tasks. We provide a brief comparison of meta-predict strategies in that respect.

Unlike all the other strategies, which are deterministic and advocate one given ranking of algorithms, the **Random** strategy has a large variance. At the first step, for instance, the variance of $LC(1)$ is $p_1 q_1$. To beat this variance, one would need to repeat random search many times, which defeats the purpose because after trying all n algorithms one is certain of finding the best one. Hence, although it may perform well on average, it is only useful as a theoretical baseline.

The three other strategies differ greatly in computational complexity at meta-training time:



(a) Artificial (b) AutoML

(c) Statlog (d) OpenML

Figure 6.17 **Learning curves on benchmark datasets.** The shaded areas represent quantiles for the Random strategy.

— **Optimal:** To determine the optimal order of algorithms of the Optimal strategy, we need to conduct a search in a search space of $n!$ permutations, to find the permutation that maximize $ALC(n)$. For each permutation, we need to evaluate n conditional probabilities.

— **Greedy:** In contrast, the Greedy strategy evaluates only $n(n+1)/2$ conditional probabilities. The Greedy+ strategy requires repeating the search n times.

— **Mean:** The Mean strategy is much faster, since it requires only evaluation n marginal probabilities.

Conditional probabilities are estimated with fewer samples than marginal probabilities. This can yield to uncertainty in algorithm ranking for the Random and Optimal strategies. While Mean has n examples to evaluate all $P(R_j = 1)$. The number of examples decays exponentially with the number of conditions to evaluate $P(R_{t_i} = 1 | R_{t_1} = 0, R_{t_2} = 0, \ldots, R_{t_{i-1}} = 0)$.

6.4.6 . Discussion

Meta-learning as an algorithm recommendation problem is, to some extent, what every overview paper is doing: Analyzing results on past tasks, the authors generally attempt to rank algorithms in order of preference, such that readers would save time by trying the smallest possible number of algorithms before obtaining satisfactory results. This paper puts a formal framework around this problem and shows that, when algorithms are not independent of one another, ranking with the Mean strategy generally does not perform as well as the Greedy strategy, which in turns is often nearly optimum. We prove theoretically that if algorithms are independent of one another and have same average performance, all strategies perform at chance level. This situation is analogous the that of the NLF theorem. If they are independent but have different average performance, then the Mean strategy is optimal. If they have the same average performance, but are not mutually independent, the Mean algorithm performs at chance level, but the Greedy algorithm can potentially do better. However, seeding the Greedy algorithm properly is important. At the expense of a slightly slower meta-training time, the Greedy+ algorithm performs a lot better. Greedy algorithms make decisions that are only "locally" optimal, hence can be outperformed by the Optimal algorithm. However, they are much faster and are therefore good candidates for use in algorithm recommendation.

Further work includes moving from binary performance scores of algorithms to continuous scores, re-defining in this context the complexity of meta-learning, and studying the final sample case. Other extensions could be done to address the problem of missing data, and the problem of "warm starting" recommendation using algorithm meta-features (a *first-order* meta-learning problem). However, at least qualitatively, our binary *zero-order* meta-learning problem setting captures the essence of meta-learning as a recommendation problem, allowing us to sort out when and how meta-learning is possible.

6.5 . Conclusion

In this chapter, we introduced three separate but related works, all of which in the context of meta-learning. Before introducing these works, we first came up with a fairly general reinforcement learning formulation of meta-learning, based on a taxonomy according their SARI tags. To our best knowledge, existing works (in the literature or proposed by us) can be comprehensively classified under this taxonomy.

Then the first work we introduced is the design and baseline results of one of our competitions in meta-learning: MetaDL challenge. This challenge is the first of a series in meta-learning. Few-shot learning problem is mainly considered here. We designed an API flexible enough to be reused for future challenges. Baseline methods such as fo-MAML and Prototypical Networks were implemented. Performance results of one baseline (fo-MAML) are presented, which confirm the advantage of meta-learning against approaches with no meta-learning at all. The challenge was accepted for the AAAI 2021 conference and a workshop will be organized on February 2021.

The second work we presented was a concrete application to power systems. We showed both theoretically and empirically that our proposed LEAP nets were capable of "super-generalization", which means that the neural network is trained on a dataset generated with certain variables $\boldsymbol{\tau}$ but can generalize on datasets generated by $\boldsymbol{\tau}$ of completely different values. This gives an interesting and somewhat extreme example of what meta-learning (and transfer learning in particular) can achieve: one can find one algorithm that generalizes even on unseen data, with very little training or no training at all. This fact gives an excellent footnote on how meta-learning can be powerful.

The third work we introduced in this chapter was our first attempt on the theoretical analysis of a very simple case of meta-learning. We showed mathematically what the optimal meta-learning strategy can be under certain conditions. In our case, only zero-order meta-learning is considered and the performances are assumed to be generated in a very simplistic way. We hope that our attempts can motivate the community to make further research on the theoretical aspect of meta-

learning and inspire a standard setting and formulation in the near future.

Method	Reference	Description	State	Action	Reward
ActivMetaL	Sun-Hosoya (2019)	For a new task, try promising learning algorithms one by one suggested by matrix factorization of the DA matrix,	S0	A1	$RC\|RP,RE$
MAML	Finn et al. (2017)	Adjust shared weights based on average validation score over all meta-train tasks and by summing gradients	S2	A1	$RE\|RP,RC$
Prototypical networks	Snell et al. (2017)	Find a common embedding and a prototype (centroid) for each class and classify examples according to distances to these prototypes	S2	A1	$RE\|RP,RC$
Auto-Sklearn (meta-learning step)	Feurer et al. (2015)	Pre-compute good hyperparameters / learning algorithm for known datasets with meta-features, recommend learning algorithms based on new task's meta-features	S1	A1	$RC\|RP,RE$
MOSAIC (meta-learning step)	Rakotoarison et al. (2019)	Pre-compute best 25 hyperparameter configurations for known datasets with selected meta-features, recommend learning algorithms based on new task's meta-features	S1	A1	$RC\|RP,RE$
Instance spaces	Muñoz et al. (2018)	Meta-features are computed and selected for datasets, based on which performance predictors are trained then applied for algorithm selection	S1	A1	$RC\|RP,RE$
Case study	Aha (1992)	Compute and select meta-features then fit performance model on artificially generated tasks for algorithm selection	S1	A1	$RC\|RP,RE$
Hyperparameter importance	Hutter et al. (2014) Van Rijn and Hutter (2018)	Use ANOVA to find most important hyperparameters of algorithms then suggest good configurations based on hyperparameters' importance	S2	A1	$RC\|RP,RE$
LEAP nets	Donnot et al. (2019)	Identify the meta-feature ($\boldsymbol{\tau}$) of the generative model and plug in the learning algorithms to be able to train on a few $\boldsymbol{\tau}$ but generalize on datasets with unseen $\boldsymbol{\tau}$ (details in Section 6.3)	S1	A0	$RC\|RP,RE$

Table 6.3 **Accuracy of baseline methods** evaluated on 600 episodes in meta-test set of Omniglot dataset. The inner learning rate of fo-MAML is . Comparison with the performance of a fully connected neural network with no meta-learning. We see that meta-learning extremely helps to improve the accuracy in the few-shot setting indeed.

Inner learning rate α	**Accuracy fo-MAML**	**Accuracy Fully Connected**
0.10	0.58 ± 0.12	0.30 ± 0.10
0.12	0.61 ± 0.13	0.28 ± 0.08
0.15	0.53 ± 0.12	0.25 ± 0.06
0.20	0.44 ± 0.16	0.24 ± 0.06
0.30	0.33 ± 0.14	0.27 ± 0.07

Table 6.4 **Learning generalization for MT, DA and LEAP:** We compare the type of generalization for Multi-task Learning (MT), Domain Adaptation (DA) and the LEAP framework. DA can learn from unlabeled examples. LEAP is capable of generalization without having seen any example (zero shot learning) on combinations of transformations, once supervised learning was performed on unary transformations.

Transfer Method	Data Set	Source Domain Reference config. $\tau = [0,0,0]$	Target Domains Unary Transform. $\tau \in \{[1,0,0], [0,1,0], [0,0,1]\}$	Target Domains Combined Transform. $\tau \in \{[1,1,0], [0,1,1], [1,0,1], [1,1,1]\}$
MT	Train	Labeled	Labeled	NA
	Test	Generalization	Generalization	NA
DA	Train	Labeled	Few or no labeled data	NA
	Test	Generalization	(Super-)generalization	NA
LEAP	Train	Labeled	Labeled	**No example**
	Test	Generalization	Generalization	**Super-generalization**

7 - Conclusion and Lessons Learned

This thesis concerns the problem of AutoML, focusing more particularly on Automated Deep Learning (AutoDL), which is a discipline in its infancy. Our contributions are both theoretical and practical, and include the organization of two challenges, a formal framework positioning the approaches in AutoML and AutoDL, novel algorithms, and their theoretical analyses.

The organization of two challenges has punctuated our work and both motivated and illustrated the formal framework introduced in Chapter 3: First, the AutoDL challenge (Chapter 4) has been a footstep of this thesis by allowing us to identify problem bottlenecks. Second, the design of the MetaDL challenge can be considered the end product of this thesis, handing over the problem to a new generation of researchers.

The formal framework (Chapter 3) structuring the AutoML and AutoDL problematic allowed us to understand and regroup contributions found in the literature and our own. Specifically, algorithms are categorized into three levels: α-level, β-level and γ-level. An α-level algorithm (called a predictor) performs only predictions (no training), a β-level algorithm (called a learner) is trained to return an α-level algorithm, a γ-level algorithm (called a meta-learner) is meta-trained to return a β-level algorithm.

In particular, the AutoDL challenge can be thought of as a β-level challenge, in the sense that participants had to submit β-level algorithms, including a train and test method, but no meta-train method. Still, in the perspective of the organizers the AutoDL challenge can also be considered as a γ-level challenge, because we provided sample tasks resembling the test tasks, which could be used for meta-learning outside the platform. Meta-learning on the platform itself was then addressed in a subsequent challenge, the MetaDL challenge (Chapter 6). Thus the MetaDL challenge is a true γ-level challenge.

The lessons learned from the AutoDL challenges, discussed in Chapter 4, shed light on the difficulties that a practitioner has to deal with,

and that we also had to face, including: (1) Solving the cross-domain AutoDL problem in a generic way, rather than injecting domain-specific knowledge; and (2) Finding effective methods returning good solutions with parsimonious computational resources. Regarding the first point, it is fair to say that the competitors made no serious effort to find a generic solution: despite the fact that all data were formatted in a uniform way, it was possible to guess the domain and naturally, they exploited it, because of the competitive nature of challenges. This resulted, in particular, in exploiting pre-trained networks or existing backbone architectures. However, in our own work, we tackled the problem of creating generic *de novo* cross-domain solutions (Chapter 5). Regarding the second point, the importance of clever engineering, and particularly effective data loading, was revealed by the challenge. This stresses the complementarity of basic research and engineering in solving real world problems. This thesis focused more on basic research.

The AutoDL challenge analysis, published in (Liu et al., 2021), motivated several contributions related to neural architecture search and meta-learning:

Neural architecture search (NAS). Formally, any hyper-parameter and architecture search problem has three ingredients: search space, search strategy, and performance evaluation method (Elsken et al., 2019). Much work has been put into improving on search strategy by importing classical optimization techniques (Boyd et al., 2004) in ML, and in performance evaluation by leveraging learning theory (Vapnik, 2013). In contrast, the definition of search space has received less attention and to our best knowledge is limited to graphs of various kinds in the literature. Indeed new toolkits such as e.g., TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al.) use a more expressive language. Still, the grammar underlying this language is not used directly as search space. The GramNAS approach proposed in this thesis takes advantage explicitly of the description of the search space, hypothesizing that this should allow us to find more powerful architectures. Initial explorations are promising, but an important limitation is the extensive computational cost of GramNAS, comparared to approaches based on graph

search. Future work includes using meta-learning strategies to initialize the search of GramNAS.

Meta-learning. Our review of the state of the art establishes that the literature abounds of meta-learning approaches; however they all tackle different tasks, making it hard to compare with one another. We thus proposed in Chapter 6 a taxonomy of meta-learning solutions complementing the taxonomy of problems proposed in Chapter 3. This classification is organized along 3 dimensions:

1. Search space "S": S0=score matrix; S1=score matrix+meta-features; S2=all datasets and algorithms
2. Actions "A" taken by meta-learning γ-level algorithms: A0=choose an α-level algorithm; A1=choose a β-level algorithm.
3. Reward "R" improvement goal(s): RP=higher *Performance*; RE = lesser number of training *Examples*; RC=lesser *Computational* resources (RC).

Three settings have been explored in more depth:

— A few-shot learning setting is inspired by MAML (Finn et al., 2017), which is in the proposed taxonomy an $(S2, A1, RE|RP, RC)$ scenario. The challenge we designed on this topic is on-going. [1]

— A super-generalization setting is motivated by a real-world application to predict flows of electricity in power lines (Donon et al., 2020a). In our taxonomy, this is a $(S1, A0, RC|RP, RE)$ scenario. The results show that super-generalization is achieved in the application setting of interest, namely, one can train a neural network to predict power flows for given power grid architectures and generalize to other grid architectures (unseen during training), without retraining the network, just by inputting the meta-feature corresponding of to the novel grid architecture. Theoretical results have corroborated experimental results, under some hypotheses.

— An active meta-learning setting, developed previously in (Sun-Hosoya, 2019), is theoretically analyzed. In our taxonomy, this

1. The results, hopefully known by the thesis defense, will be presented then.

is a $(S0, A1, RC|RP, RE)$ scenario. Given a score matrix R_{ij} , we demonstrated that NFL is indeed an intrinsic limitation when R_{ij} is drawn i.i.d from a given mother distribution: the average performance of the random search cannot be beaten. However, when there are dependencies between lines and/or columns, the NLF does not apply.

The legacy and perspective of this thesis includes:

— **Datasets.** A repository of around 100 datasets was made available in a tensor format (with much effort). 26 of these datasets were already published along with the AutoDL challenge [2]. More datasets will soon be made public after the organization of future challenges. The community is invited to utilize this repository to push forward meta-learning research;

— **Toolkits.** Python toolkits and packages for formatting datasets have been developed and open-sourced [3]. This provides the community a convenient tool to format their own data in the same format and apply winning solutions and related approaches easily;

— **Open-sourced winning solutions.** Winning solutions of the AutoDL challenges are open-sourced too [4] and the anyone can run them very easily, especially on datasets in the above repository and those customizedly formatted. Also, the fact sheets describing more details of each winning method were collected and published after each challenge (Liu et al., 2021);

— **Meta-dataset and benchmark.** The AutoDL challenges provide a natural meta-dataset in its own nature. We released this meta-dataset in our TPAMI paper (Liu et al., 2021) and on the website https://autodl.chalearn.org/benchmark. We note that AutoDL challenges also provide an extensible benchmark since most of the challenges are still open for solutions (although no prizes are provided). We also submitted a paper (Ullah et al., 2021) on the

2. https://autodl.lri.fr/competitions/162#learn_the_details-get_data
3. https://github.com/zhengying-liu/autodl-contrib
4. https://autodl.chalearn.org/

datasets of the MetaDL challenge to the NeurIPS 2021 Datasets and Benchmarks Track.

In retrospect, the organization of a challenge offers a PhD student a rare opportunity to define and tackle a research question, identify their factors of difficulty, and ultimately leverage the help of the whole community of participants to solve this question, while enforcing reproducible research. However, this is very time consuming, and leveraging challenges or benchmarks organized by others may also be another way to conduct sound experimental validations. The legacy of this thesis should facilitate the work of the next generation of PhD students.

Many directions of research we have started exploring could be pursued using the benchmarks we created. For one, the possibilities of GramNAS remained under-exploited. As discussed in Chapter 5, one could warm start the search part of the Monte-Carlo Tree Search by using a hierarchy of formal grammars. In Chapter 3 and 6 we started devising a theory of meta-learning. Much remains to be done, in particular mitigating over-fitting at the meta-learning level, computing performance bounds, and deriving regularized meta-learning algorithms. We started doing work in this direction. While the MetaDL challenge offers a benchmark of any-level of meta-learning, our theory is so-far limited to zero-order (meta-learning solely from a performance matrix). Extending meta-learning theory to first and second order meta-learning remains to be done. Finally, when the MetaDL challenge terminates, its analysis reveals new interesting results and kick-start new research avenues.

Bibliography

CodaLab - Home. URL https://competitions.codalab.org/.

The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific Data*, 5 (1):180161, 2018.

Spearman's rank correlation coefficient, Apr. 2020a. URL https://en.wikipedia.org/w/index.php?title=Spearman%27s_rank_correlation_coefficient&oldid=953109044. Page Version ID: 953109044.

Unicode, Aug. 2020b. URL https://en.wikipedia.org/w/index.php?title=Unicode&oldid=971117701. Page Version ID: 971117701.

Backus–Naur form, June 2020a. Page Version ID: 964894134.

Extended Backus–Naur form, June 2020b. Page Version ID: 964573381.

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

D. W. Aha. Generalizing from case studies: A case study. In *Machine Learning Proceedings 1992*, pages 1–10. Elsevier, 1992.

E. I. Albert Bifet. Airlines dataset inspired in the regression dataset from elena ikonomovska. the task is to predict whether a given flight will be delayed, given the information of the scheduled departure., 2009. URL http://kt.ijs.si/elena_ikonomovska/data.html.

O. Alsac and B. Stott. Optimal load flow with steady-state security. *IEEE transactions on power apparatus and systems*, PAS-93(3):745–751, 1974.

K. Arulkumaran, A. Cully, and J. Togelius. Alphastar: An evolutionary computation perspective. *CoRR*, abs/1902.01724, 2019. URL http://arxiv.org/abs/1902.01724.

F. Assunção, N. Lourenço, B. Ribeiro, and P. Machado. Evolution of scikit-learn pipelines with dynamic structured grammatical evolution. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 530–545. Springer, 2020.

P. Auer. Using confidence bounds for exploitation-exploration tradeoffs. *The Journal of Machine Learning Research*, 3(null):397–422, Mar. 2003. ISSN 1532-4435.

D. Auger, A. Couetoux, and O. Teytaud. Continuous upper confidence trees with polynomial exploration–consistency. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 194–209. Springer, 2013.

S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv:1803.01271 [cs]*, Apr. 2018. URL http://arxiv.org/abs/1803.01271. arXiv: 1803.01271.

B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing Neural Network Architectures using Reinforcement Learning. *arXiv:1611.02167 [cs]*, Nov. 2016. URL http://arxiv.org/abs/1611.02167. arXiv: 1611.02167.

B. Baker, O. Gupta, N. Naik, and R. Raskar. DESIGNING NEURAL NETWORK ARCHITECTURES USING REINFORCEMENT LEARNING. page 18, 2017.

A. E. Baz, I. Guyon, Z. Liu, J. N. v. Rijn, S. Treguer, and J. Vanschoren. MetaDL challenge design and baseline results. In *AAAI Workshop on Meta-Learning and MetaDL Challenge*, pages 1–16. PMLR, Aug. 2021. URL https://proceedings.mlr.press/v140/el-baz21a.html. ISSN: 2640-3498.

Y. Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep): 1089–1105, 2004.

K. P. Bennett, J. Hu, X. Ji, G. Kunapuli, and J.-S. Pang. Model selection via bilevel optimization. In *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, pages 1922–1929. IEEE, 2006.

J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012. ISSN ISSN 1533-7928. URL http://www.jmlr.org/papers/v13/bergstra12a.html.

J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013.

J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554.

Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf.

C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, July 2013. ISSN 0950-7051. doi: 10.1016/j.knosys.2013.03.012. URL http://www.sciencedirect.com/science/article/pii/S0950705113001044.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM. ISBN 978-0-89791-497-0. doi: 10.1145/130385.130401. URL http://doi.acm.org/10.1145/130385.130401. event-place: Pittsburgh, Pennsylvania, USA.

S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

K. Brain. AutoCLINT, Automatic Computationally LIght Network Transfer. https://github.com/kakaobrain/autoclint, 2019.

P. Brazdil, C. G. Carrier, C. Soares, and R. Vilalta. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

J. S. Bridle and M. D. Brown. An experimental automatic word recognition system. *JSRU Report*, 1003(5):33, 1974.

N. Brown and T. Sandholm. Safe and nested subgame solving for imperfect-information games. In *Advances in neural information processing systems*, pages 689–699, 2017.

S. Bubeck and N. Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Efficient Architecture Search by Network Transformation. *arXiv:1707.04873 [cs]*, July 2017. URL http://arxiv.org/abs/1707.04873. arXiv: 1707.04873.

H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. Dec. 2018. URL http://arxiv.org/abs/1812.00332. arXiv: 1812.00332.

R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Twenty-first international conference on Machine learning - ICML '04*, page 18, Banff, Alberta, Canada, 2004. ACM Press. doi: 10.1145/1015330.1015432. URL http://portal.acm.org/citation.cfm?doid=1015330.1015432.

G. Chaslot, S. Bakkes, I. Szita, and P. Spronck. Monte-carlo tree search: A new framework for game ai. 2008.

X. Chen, Q. Lin, and ... Neural feature search: A neural architecture for automated feature engineering. In *ICDM'19*, October 2019.

X. Chen, L. Xie, J. Wu, and Q. Tian. Progressive darts: Bridging the optimization gap for nas in the wild. *International Journal of Computer Vision*, 129(3):638–655, 2021.

K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

N. Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

X. Chu, B. Zhang, R. Xu, and J. Li. FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. *arXiv:1907.01845 [cs, stat]*, Mar. 2020. URL http://arxiv.org/abs/1907.01845. arXiv: 1907.01845.

J. S. Chung, A. Nagrani, and A. Zisserman. Voxceleb2: Deep speaker recognition. In *INTERSPEECH*, 2018.

C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang. AdaNet: Adaptive Structural Learning of Artificial Neural Networks. Feb. 2017. URL http://arxiv.org/abs/1607.01097. arXiv: 1607.01097.

N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005. doi: 10.1109/CVPR.2005.177.

N. Das, S. Ghosh, T. Gonçalves, and P. Quaresma. Comparison of different graph distance metrics for semantic text based classification. *Polibits*, (49):51–58, 2014.

S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4):357–366, 1980.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Oct. 2018. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

T. T. H. Dinh, T. H. Chu, and Q. U. Nguyen. Transfer learning in genetic programming. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1145–1151. IEEE, 2015.

X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://openreview.net/forum?id=HJxyZkBKDr.

B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici. Fast power system security analysis with guided dropout. In *ESANN*, Apr. 2018a. URL https://hal.archives-ouvertes.fr/hal-01695793.

B. Donnot, I. Guyon, M. Schoenauer, A. Marot, and P. Panciatici. Anticipating contingencies in power grids using fast neural net screening. In *IEEE WCCI 2018*, Rio de Janeiro, Brazil, July 2018b. URL https://hal.archives-ouvertes.fr/hal-01783669.

B. Donnot, B. Donon, I. Guyon, Z. Liu, A. Marot, P. Panciatici, and M. Schoenauer. LEAP nets for power grid perturbations. *arXiv:1908.08314 [cs, eess, stat]*, Aug. 2019. URL http://arxiv.org/abs/1908.08314. arXiv: 1908.08314.

B. Donon, B. Donnot, I. Guyon, Z. Liu, A. Marot, P. Panciatici, and M. Schoenauer. Leap nets for system identification and application to power systems. *Neurocomputing*, 2020a.

B. Donon, B. Donnot, I. Guyon, Z. Liu, A. Marot, P. Panciatici, and M. Schoenauer. LEAP nets for system identification and application to power systems. *Neurocomputing*, 416:316–327, Nov. 2020b. ISSN 0925-2312. doi: 10.1016/j.neucom.2019.12.135. URL https://www.sciencedirect.com/science/article/pii/S0925231220305051.

B. Donon, Z. Liu, W. LIU, I. Guyon, A. Marot, and M. Schoenauer. Deep Statistical Solvers. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7910–7921. Curran Associates, Inc., 2020c. URL https://proceedings.neurips.cc/paper/2020/file/5a16bce575f3ddce9c819de125ba0029-Paper.pdf.

M. M. Drugan and A. Nowe. Designing multi-objective multi-armed bandits algorithms: A study. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2013.

R. Egele, P. Balaprakash, V. Vishwanath, I. Guyon, and Z. Liu. AgEBO-Tabular: Joint Neural Architecture and Hyperparameter Search with Autotuned Data-Parallel Training for Tabular Data. *arXiv:2010.16358 [cs, stat]*, Oct. 2021. URL http://arxiv.org/abs/2010.16358. arXiv: 2010.16358.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20:55:1–55:21, 2019.

N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

H. J. Escalante, M. Montes, and L. E. Sucar. Particle Swarm Model Selection. *Journal of Machine Learning Research*, 10(Feb):405–440, 2009. ISSN ISSN 1533-7928. URL http://www.jmlr.org/papers/v10/escalante09a.html.

S. Estevez-Velarde, Y. Gutiérrez, A. Montoyo, and Y. Almeida-Cruz. Automl strategy based on grammatical evolution: A case study about knowledge discovery from text. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4356–4365, 2019.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. page 10.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and Robust Automated Machine Learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.

M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Practical automated machine learning for the automl challenge 2018. In *AutoML workshop at international conference on machine learning (ICML)*, 2018.

C. Finn, P. Abbeel, and S. Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. Mar. 2017. URL http://arxiv.org/abs/1703.03400. arXiv: 1703.03400.

C. Finn, A. Rajeswaran, S. Kakade, and S. Levine. Online Meta-Learning. Feb. 2019. URL http://arxiv.org/abs/1902.08438. arXiv: 1902.08438.

L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning*, pages 1568–1577, July 2018. URL http://proceedings.mlr.press/v80/franceschi18a.html.

N. Fusi, R. Sheth, and H. M. Elibol. Probabilistic Matrix Factorization for Automated Machine Learning. May 2018. URL http://arxiv.org/abs/1705.05355. arXiv: 1705.05355.

A. Geyer-Schulz. *Fuzzy Rule-based Expert Systems and Genetic Machine Learning*. Number v. 3 in Fuzzy Rule-based Expert Systems and Genetic Machine Learning. Physica-Verlag, 1995. ISBN 978-3-7908-0830-8. URL https://books.google.lu/books?id=yoFQAAAAMAAJ.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016. ISBN 0262035618, 9780262035613.

I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *arXiv:1406.2661 [cs, stat]*, June 2014. URL http://arxiv.org/abs/1406.2661. arXiv: 1406.2661.

A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013. doi: 10.1109/ICASSP.2013.6638947. ISSN: 2379-190X.

A. Graves, G. Wayne, and I. Danihelka. Neural Turing Machines. *arXiv:1410.5401 [cs]*, Dec. 2014. URL http://arxiv.org/abs/1410.5401. arXiv: 1410.5401.

I. Guyon, G. Cawley, and G. Dror. *Hands-On Pattern Recognition: Challenges in Machine Learning, Volume 1*. Microtome Publishing, Brookline, MA, USA, 2011. ISBN 0971977712.

I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, Tin Kam Ho, N. Macia, B. Ray, M. Saeed, A. Statnikov, and E. Viegas. Design of the 2015 ChaLearn AutoML challenge. pages 1–8. IEEE, July 2015. ISBN 978-1-4799-1960-4. doi: 10.1109/IJCNN.2015.7280767. URL http://ieeexplore.ieee.org/document/7280767/.

I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag, A. Statnikov, W.-W. Tu, and E. Viegas. Analysis of the AutoML Challenge series 2015-2018. In F. Hutter, L. Kotthoff, and J. Vanschoren, editors, *AutoML: Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning. Springer Verlag, 2018. URL https://hal.archives-ouvertes.fr/hal-01906197.

W. Han, M. Xu, J. Zhang, C. Wang, K. Zhang, and X. S. Wang. Transpcfg: Transferring the grammars from short passwords to guess long passwords effectively. *IEEE Transactions on Information Forensics and Security*, 16:451–465, 2020.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. Dec. 2015. URL http://arxiv.org/abs/1512.03385. arXiv: 1512.03385.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, pages 630–645. Springer, 2016.

D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949. ISBN 0-8058-4300-0.

P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification, 2017.

S. Hettich and S. D. Bay. The uci kdd archive, 1999. URL http://kdd.ics.uci.edu.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

T. Hossen, S. J. Plathottam, R. K. Angamuthu, P. Ranganathan, and H. Salehfar. Short-term load forecasting using deep neural networks (dnn). In *2017 North American Power Symposium (NAPS)*, pages 1–6, Sept 2017. doi: 10.1109/NAPS.2017.8107271.

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. pages 507–523.

F. Hutter, D. A. Tompkins, and H. H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In *International Conference on Principles and Practice of Constraint Programming*, pages 233–248. Springer, 2002.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-based Optimization for General Algorithm Configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, pages 507–523, Berlin, Heidelberg, 2011a. Springer-Verlag. ISBN 978-3-642-25565-6. doi: 10.1007/978-3-642-25566-3_40. URL http://dx.doi.org/10.1007/978-3-642-25566-3_40. event-place: Rome, Italy.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011b.

F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pages 754–762. PMLR, 2014.

F. Hutter, L. Kotthoff, and J. Vanschoren, editors. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, 2018. In press, available at http://automl.org/book.

S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Feb. 2015. URL https://arxiv.org/abs/1502.03167.

H. Jin, Q. Song, and X. Hu. Auto-Keras: An Efficient Neural Architecture Search System. In *25th ACM SIGKDD International Conference*, pages 1946–1956, July 2019. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330648.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431, Valencia, Spain, Apr. 2017. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/E17-2068.

A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

J. N. Kather, C.-A. Weis, F. Bianconi, S. M. Melchers, L. R. Schad, T. Gaiser, A. Marx, and F. G. Z"ollner. Multi-class texture analysis in colorectal cancer histology. *Scientific reports*, 6:27988, 2016.

M. Katz, P. Ram, S. Sohrabi, and O. Udrea. Exploring context-free languages via planning: The case for automating machine learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 403–411, 2020.

G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://www.aclweb.org/anthology/D14-1181.

D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Dec. 2014. URL http://arxiv.org/abs/1412.6980. arXiv: 1412.6980.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536. PMLR, Apr. 2017. URL http://proceedings.mlr.press/v54/klein17a.html.

L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Germany, Sept. 2006. Springer-Verlag. ISBN 978-3-540-45375-8. doi: 10.1007/11871842_29. URL https://doi.org/10.1007/11871842_29.

R. Kohavi. A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, pages 1137–1143, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-363-9. URL http://dl.acm.org/citation.cfm?id=1643031.1643047. event-place: Montreal, Quebec, Canada.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

K. Kumagai, I. Kobayashi, D. Mochihashi, H. Asoh, T. Nakamura, and T. Nagai. Human-like natural language generation using monte carlo tree search. In *Proceedings of the INLG 2016 Workshop on Computational Creativity in Natural Language Generation*, pages 11–18, 2016.

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266): 1332–1338, Dec. 2015. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aab3050. URL http://www.sciencemag.org/cgi/doi/10.1126/science.aab3050.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, Nov. 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436, May 2015. URL http://dx.doi.org/10.1038/nature14539.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *arXiv:1603.06560 [cs, stat]*, Mar. 2016. URL http://arxiv.org/abs/1603.06560. arXiv: 1603.06560.

S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim. Fast autoaugment. In *Advances in Neural Information Processing Systems*, pages 6662–6672, 2019a.

S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim. Fast AutoAugment. May 2019b. URL http://arxiv.org/abs/1905.00397. arXiv: 1905.00397.

M. Lindauer, H. H. Hoos, F. Hutter, and T. Schaub. AutoFolio: an automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53(1):745–778, May 2015. ISSN 1076-9757.

M. Lindauer, J. N. van Rijn, and L. Kotthoff. Open algorithm selection challenge 2017: Setup and scenarios. In *Open Algorithm Selection Challenge 2017*, pages 1–7, 2017.

M. Lindauer, J. N. van Rijn, and L. Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive Neural Architecture Search. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, volume 11205, pages 19–35. Springer International Publishing, Cham, 2018a. ISBN 978-3-030-01245-8 978-3-030-01246-5. doi: 10.1007/978-3-030-01246-5_2. URL http://link.springer.com/10.1007/978-3-030-01246-5_2.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable Architecture Search. *arXiv:1806.09055 [cs, stat]*, June 2018b. URL http://arxiv.org/abs/1806.09055. arXiv: 1806.09055.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable Architecture Search. Apr. 2019a. URL http://arxiv.org/abs/1806.09055. arXiv: 1806.09055.

Z. Liu and I. Guyon. Asymptotic analysis of meta-learning as a recommendation problem. In I. Guyon, J. N. van Rijn, S. Treguer, and J. Vanschoren, editors, *AAAI Workshop on Meta-Learning and MetaDL Challenge*, volume 140 of *Proceedings of Machine Learning Research*, pages 100–114. PMLR, 09 Feb 2021. URL https://proceedings.mlr.press/v140/liu21a.html.

Z. Liu, I. Guyon, J. J. Junior, M. Madadi, S. Escalera, A. Pavao, H. J. Escalante, W.-W. Tu, Z. Xu, and S. Treguer. AutoCV Challenge Design and Baseline Results, July 2019b. URL https://hal.archives-ouvertes.fr/hal-02265053.

Z. Liu, Z. Xu, S. Escalera, I. Guyon, J. C. S. Jacques Junior, M. Madadi, A. Pavao, S. Treguer, and W.-W. Tu. Towards Automated Computer Vision: Analysis of the AutoCV Challenges 2019. *Pattern Recognition Letters*, Apr. 2020a. ISSN 0167-8655. doi: 10.1016/j.patrec.2020.04.030. URL http://www.sciencedirect.com/science/article/pii/S0167865520301525.

Z. Liu, Z. Xu, S. Rajaa, M. Madadi, J. Julio C. S. Jacques, S. Escalera, A. Pavao, S. Treguer, W.-W. Tu, and I. Guyon. Towards Automated Deep Learning: Analysis of the AutoDL challenge series 2019. Proceedings of Machine Learning Research, 2020b.

Z. Liu, Z. Xu, S. Rajaa, M. Madadi, J. C. S. J. Junior, S. Escalera, A. Pavao, S. Treguer, W.-W. Tu, and I. Guyon. Towards Automated Deep Learning: Analysis of the AutoDL challenge series 2019. In *NeurIPS 2019 Competition and Demonstration Track*, pages 242–252. PMLR, Aug. 2020c. URL http://proceedings.mlr.press/v123/liu20a.html. ISSN: 2640-3498.

Z. Liu, A. Pavao, Z. Xu, S. Escalera, F. Ferreira, I. Guyon, S. Hong, F. Hutter, R. Ji, J. C. S. J. Junior, G. Li, M. Lindauer, Z. Luo, M. Madadi, T. Nierhoff, K. Niu, C. Pan, D. Stoll, S. Treguer, J. Wang, P. Wang, C. Wu, Y. Xiong, A. Zela, and Y. Zhang. Winning solutions and post-challenge analyses of the chalearn autodl challenge 2019. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3108–3125, 2021. doi: 10.1109/TPAMI.2021.3075372. URL https://ieeexplore.ieee.org/document/9415128.

I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL https://doi.org/10.1023/B:VISI.0000029664.99615.94.

R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural architecture optimization. *arXiv preprint arXiv:1808.07233*, 2018.

W. Mcculloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

R. I. Mckay, N. X. Hoai, P. A. Whigham, Y. Shan, and M. O'neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.

P. Mermelstein. Distance measures for speech recognition, psychological and instrumental. *Pattern Recognition and Artificial Intelligence*, pages 374–388, 1976.

A. Mezine, A. Llamosi, V. Letort, M. Sebag, and F. d'Alché Buc. Autonomous learning of parameters in differential equations. In *32nd International Conference on Machine Learning (ICML)-AutoML workshop*, 2015.

K. O. S. a. R. Miikkulainen. Evolving Neural Networks Through Augmenting Topologies. 2002. URL http://nn.cs.utexas.edu/?stanley:ec02.

M. Mısır and M. Sebag. Alors: An algorithm recommender system. *Artificial Intelligence*, 244:291–314, 2017.

T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997. ISBN 978-0-07-042807-2.

M. C. Mozer. Induction of multiscale temporal structure. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 275–282. Morgan-Kaufmann, 1992. URL http://papers.nips.cc/paper/522-induction-of-multiscale-temporal-structure.pdf.

M. A. Muñoz, L. Villanova, D. Baatar, and K. Smith-Miles. Instance spaces for machine learning classification. *Machine Learning*, 107(1):109–147, Jan. 2018. ISSN 1573-0565. doi: 10.1007/s10994-017-5629-5. URL https://doi.org/10.1007/s10994-017-5629-5.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

R. Negrinho and G. Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017a.

R. Negrinho and G. Gordon. DeepArchitect: Automatically Designing and Training Deep Architectures. *arXiv:1704.08792 [cs, stat]*, Apr. 2017b. URL http://arxiv.org/abs/1704.08792. arXiv: 1704.08792.

Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr*, volume 269, pages 543–547, 1983.

T. Nguyen. Neural network load-flow. *IEE Proceedings - Generation, Transmission and Distribution*, 142:51–58(7), January 1995. ISSN 1350-2360. URL http://digital-library.theiet.org/content/journals/10.1049/ip-gtd_19951484.

S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knoweledge and Data Engineering*, 22(10):1345–1359, October 2010.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. page 4.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. ISSN 1533-7928. URL http://jmlr.org/papers/v12/pedregosa11a.html.

D. Perez, S. Mostaghim, S. Samothrakis, and S. M. Lucas. Multiobjective monte carlo tree search for real-time games. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4):347–360, 2014.

H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient Neural Architecture Search via Parameter Sharing. Feb. 2018. URL http://arxiv.org/abs/1802.03268. arXiv: 1802.03268.

J. Pineau, P. Vincent-Lamarre, K. Sinha, V. Larivière, A. Beygelzimer, F. d'Alché Buc, E. Fox, and H. Larochelle. Improving reproducibility in machine learning research (a report from the neurips 2019 reproducibility program). *arXiv preprint arXiv:2003.12206*, 2020.

H. Rakotoarison, M. Schoenauer, and M. Sebag. Automated machine learning with monte-carlo tree search.

H. Rakotoarison, M. Schoenauer, and M. Sebag. Automated Machine Learning with Monte-Carlo Tree Search. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 3296–3303, Macao, China, Aug. 2019. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-0-9992411-4-1. doi: 10.24963/ijcai.2019/457. URL https://www.ijcai.org/proceedings/2019/457.

E. Rasmusen. *Games and information, an introduction to game theory*. Rasmusen, Erasmuse@ indiana. edu., 2005.

A. Ratle and M. Sebag. Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In *International Conference on Parallel Problem Solving from Nature*, pages 211–220. Springer, 2000.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. 2016.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-Scale Evolution of Image Classifiers. Mar. 2017. URL http://arxiv.org/abs/1703.01041. arXiv: 1703.01041.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789, July 2019. ISSN 2374-3468. doi: 10.1609/aaai.v33i01.33014780. URL https://www.aaai.org/ojs/index.php/AAAI/article/view/4405. Number: 01.

E. Real, C. Liang, D. R. So, and Q. V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. *arXiv:2003.03384 [cs, stat]*, Mar. 2020. URL http://arxiv.org/abs/2003.03384. arXiv: 2003.03384.

P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

J. R. Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, Oct. 1986a. doi: 10.1038/323533a0. URL http://adsabs.harvard.edu/abs/1986Natur.323..533R.

D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, editors. *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*. MIT Press, Cambridge, MA, USA, 1986b. ISBN 978-0-262-68053-0.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *arXiv:1409.0575 [cs]*, Jan. 2015. URL http://arxiv.org/abs/1409.0575. arXiv: 1409.0575.

S. Russell and P. Norvig. Artificial intelligence: a modern approach. 2002.

J. Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987. URL http://www.idsia.ch/~juergen/diploma.html.

R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, 2019.

M. Sipser. *Introduction to the theory of computation*. Boston : PWS Pub. Co., 1997. URL http://archive.org/details/introductiontoth00sips.

J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2951–2959. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. M. A. Patwary, Prabhat, and R. P. Adams. Scalable Bayesian Optimization Using Deep Neural Networks. *arXiv:1502.05700 [stat]*, Feb. 2015. URL http://arxiv.org/abs/1502.05700. arXiv: 1502.05700.

N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

L. Sun-Hosoya. *Meta-Learning as a Markov Decision Process*. phdthesis, Université Paris Saclay (COmUE), Dec. 2019. URL https://hal.archives-ouvertes.fr/tel-02422144.

L. Sun-Hosoya, I. Guyon, and M. Sebag. *ActivMetaL: Algorithm Recommendation with Active Meta Learning*. Sept. 2018a. URL https://hal.archives-ouvertes.fr/hal-01931262. Published: IAL 2018 workshop, ECML PKDD.

L. Sun-Hosoya, I. Guyon, and M. Sebag. Activmetal: Algorithm recommendation with active meta learning. In *IAL 2018 workshop, ECML PKDD*, 2018b.

F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015. doi: 10.1109/CVPR.2015.7298594.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.308. URL http://ieeexplore.ieee.org/document/7780677/.

M. Tan and Q. V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Nov. 2019. URL http://arxiv.org/abs/1905.11946. arXiv: 1905.11946.

O. Teytaud, S. Gelly, and M. Sebag. Anytime many-armed bandits. 2007.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms. *arXiv:1208.3719 [cs]*, Aug. 2012. URL http://arxiv.org/abs/1208.3719. arXiv: 1208.3719.

D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, and H. Larochelle. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. *arXiv:1903.03096 [cs, stat]*, Oct. 2019. URL http://arxiv.org/abs/1903.03096. arXiv: 1903.03096.

I. Ullah, P. A. Vu, H. Sun, M. Huisman, F. Mohr, J. N. van Rijn, A. Plaat, A. E. Baz, Z. Liu, J. Vanschoren, and I. Guyon. Meta-Album: Multi-domain Benchmark for Few-Shot Image Classification. *Under review for NeurIPS 2021 Track Datasets and Benchmarks*, page 14, 2021.

J. N. Van Rijn and F. Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2367–2376, 2018.

J. N. Van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren. Openml: A collaborative science platform. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 645–649. Springer, 2013.

F. van Veen. Fjodor van Veen, Author at The Asimov Institute, 2017. URL https://www.asimovinstitute.org/author/fjodorvanveen/.

J. Vanschoren. Meta-Learning: A Survey. *arXiv:1810.03548 [cs, stat]*, Oct. 2018. URL http://arxiv.org/abs/1810.03548. arXiv: 1810.03548.

J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15 (2):49–60, June 2014. ISSN 19310145. doi: 10.1145/2641190.2641198. URL http://arxiv.org/abs/1407.7722. arXiv: 1407.7722.

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Science & Business Media, June 2013. ISBN 978-1-4757-3264-1. Google-Books-ID: EqgACAAAQBAJ.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. June 2017. URL https://arxiv.org/abs/1706.03762v5.

R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

L. Wang, Y. Zhao, and Y. Jinnai. AlphaX: eXploring Neural Architectures with Deep Neural Networks and Monte Carlo Tree Search. *arXiv:1805.07440 [cs, stat]*, May 2018. URL http://arxiv.org/abs/1805.07440. arXiv: 1805.07440.

W. Wang and M. Sebag. Multi-objective Monte-Carlo Tree Search. page 16, 2012.

Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. *Advances in Neural Information Processing Systems*, 21: 1729–1736, 2008.

Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Computing Surveys*, 53(3):63:1–63:34, June 2020. ISSN 0360-0300. doi: 10.1145/3386252. URL https://doi.org/10.1145/3386252.

P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

P. A. Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995.

M. Wistuba, A. Rawat, and T. Pedapati. A Survey on Neural Architecture Search. *arXiv:1905.01392 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1905.01392. arXiv: 1905.01392.

D. Wolpert. The Supervised Learning No-Free-Lunch Theorems. In *Proceedings of the 6th Online World Conference on Soft Computing in Industrial Applications*, Jan. 2001. doi: 10.1007/978-1-4471-0123-9_3.

D. H. Wolpert. The Lack of A Priori Distinctions Between Learning Algorithms. *Neural Computation*, 8(7):1341–1390, Oct. 1996. ISSN 0899-7667. doi: 10.1162/neco.1996.8.7.1341. URL https://doi.org/10.1162/neco.1996.8.7.1341.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, Apr. 1997. ISSN 1089-778X. doi: 10.1109/4235.585893. URL https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf.

M. L. Wong and K. S. Leung. Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180, 1997.

L. Xie and A. Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

L. Xu, H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. pages 210–216.

L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *Proc. of RCRA workshop at IJCAI*, 2011.

L. Xu, F. Hutter, J. Shen, H. Hoos, and K. Leyton-Brown. SATzilla2012: Improved algorithm selection based on cost-sensitive classification models. *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, pages 55–58, Jan. 2012.

A. Yang, P. M. Esperança, and F. M. Carlucci. NAS evaluation is frustratingly hard. Jan. 2020. URL http://arxiv.org/abs/1912.12522. arXiv: 1912.12522.

C. Ying, A. Klein, E. Real, E. Christiansen, K. Murphy, and F. Hutter. NAS-Bench-101: Towards Reproducible Neural Architecture Search. *arXiv:1902.09635 [cs, stat]*, Feb. 2019. URL http://arxiv.org/abs/1902.09635. arXiv: 1902.09635.

A. Zela, J. Siems, and F. Hutter. NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search. *arXiv:2001.10422 [cs, stat]*, Apr. 2020. URL http://arxiv.org/abs/2001.10422. arXiv: 2001.10422.

Y. Zhang, Z. Lin, J. Jiang, Q. Zhang, Y. Wang, H. Xue, C. Zhang, and Y. Yang. Deeper Insights into Weight Sharing in Neural Architecture Search. *arXiv:2001.01431 [cs, stat]*, Jan. 2020. URL http://arxiv.org/abs/2001.01431. arXiv: 2001.01431.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-pytorch tabular: Multi-fidelity metalearning for efficient and robust autodl. *arXiv preprint arXiv:2006.13799*, 2020.

R. D. Zimmerman and et al. Matpower. *IEEE Trans. on Power Systems*, pages 12–19, 2011.

B. Zoph and Q. V. Le. Neural Architecture Search with Reinforcement Learning. *arXiv:1611.01578 [cs]*, Nov. 2016. URL http://arxiv.org/abs/1611.01578. arXiv: 1611.01578.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning Transferable Architectures for Scalable Image Recognition. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8697–8710, Salt Lake City, UT, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00907. URL https://ieeexplore.ieee.org/document/8579005/.

K. J. Åström. Optimal Control of Markov Processes with Incomplete State Information I. *Journal of Mathematical Analysis and Applications*, 10:174–205, 1965. ISSN 0022-247X. URL http://lup.lub.lu.se/record/8867084. Publisher: Elsevier.

A - Toy Example on Real Number Approximation with GramNAS

To have a proof-of-concept and also show that the approach GramNAS is not specific to NAS, we first applied it to the problem of real number approximation. In this problem, a real number $x^* \in \mathbb{R}$ is chosen by the system in advance and the agent can make guesses to approximate this number as much as possible. If the agent makes a guess $x \in \mathbb{R}$, then a reward

$$r = -\log_{10}|x - x^*| \tag{A.1}$$

is given to the agent as feedback. So the closer x is to x^*, the more reward the agent will get.

Data. There is no dataset in this problem. The only data is the ground truth x^* which is chosen to be $1/\alpha \approx 137.035999139$ [1] in our experiments.

Grammar. The search space in this problem is the set of all real numbers $\mathbb{R}$. We will consider all decimal numbers instead for practical reasons. The grammar we use is similar to the one on floating numbers we saw in a previous section and we recall it as follows (note that this time we only consider positive numbers).

```
start: decimal

decimal: integer "." integer

integer: digit integer
       | digit

digit: "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
```

In this grammar, there are two rules starting from *integer*, namely either add a *digit* then continue or become a *digit* then stop. Thus the search-

1. Related to the fine-structure constant α in physics. We have also run similar experiments for $\pi \approx 3.14159265358...$ and got similar results.

ing agent needs to make this decision each time when it encounters the non-terminal *integer*.

Baseline. The baseline we use is *random search within the grammar*, which just repetitively samples random words using the method introduced in 5.2.5 (with $\beta = 0.1$).

Results. We run both approach for 1000 iterations. The UCB constant for MCTS is set to 0.1. We plot the reward of the best guess so far vs number of iterations, as shown in Figure A.1. We see that our ap-

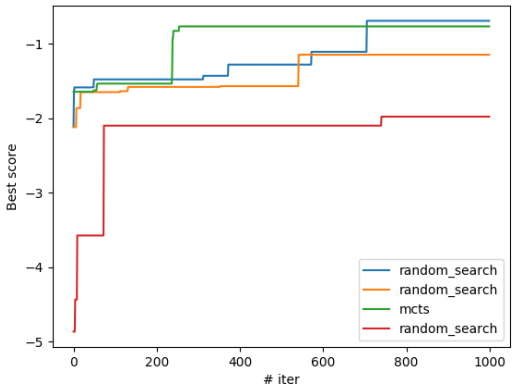


Figure A.1 GramNAS (*mcts*) vs random search within grammar (*random_search*). Best score in y-axis is the maximum value of $\log_{10}|x_{i^*} - x^*|$ for the best guess x_{i^*} so far.

proach (the green line) finds good approximation (with a difference less than 10 to the ground truth) after around 200 iterations, consistently superior to random search. Firstly, this result validates our approach of grammar + MCTS since the curve consistently goes up and continues to explore better approximation. Secondly, this result and also shows the effectiveness of MCTS since the two methods only differ in search strategy.

B - Papers published during this thesis

For a documentary purpose, we list papers that are co-authored by the author of this thesis and were published or prepared during this PhD. These papers were attached in appendix to facilitate the reviewing process. For the final version of this thesis though, we only provide pointers to them to avoid possible issues. These papers are:

— **AutoDL post-challenge analysis paper.** The AutoDL post-challenge analysis paper was published in *IEEE Transactions on Pattern Analysis and Machine Intelligence* (TPAMI):
Z. Liu, A. Pavao, Z. Xu, S. Escalera, F. Ferreira, I. Guyon, S. Hong, F. Hutter, R. Ji, J. C. S. J. Junior, G. Li, M. Lindauer, Z. Luo, M. Madadi, T. Nierhoff, K. Niu, C. Pan, D. Stoll, S. Treguer, J. Wang, P. Wang, C. Wu, Y. Xiong, A. Zela, and Y. Zhang. Winning solutions and post-challenge analyses of the chalearn autodl challenge 2019. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9): 3108–3125, 2021. doi: 10.1109/TPAMI.2021.3075372. URL https://ieeexplore.ieee.org/document/9415128.

— **AutoCV & AutoCV2 analysis paper.** The analysis paper on AutoCV challenge and AutoCV2 challenge, the first two challenges in the AutoDL challenge series was published in *Elsevier Pattern Recognition Letters* (PRL):
Z. Liu, Z. Xu, S. Escalera, I. Guyon, J. C. S. Jacques Junior, M. Madadi, A. Pavao, S. Treguer, and W.-W. Tu. Towards Automated Computer Vision: Analysis of the AutoCV Challenges 2019. *Pattern Recognition Letters*, Apr. 2020a. ISSN 0167-8655. doi: 10.1016/j.patrec.2020.04.030. URL http://www.sciencedirect.com/science/article/pii/S0167865520301525

— **AutoNLP & AutoSpeech analysis paper.** The analysis paper on AutoNLP challenge and AutoSpeech challenge, the third and fourth challenges in the AutoDL challenge series, was published in *Proceedings of Machine Learning Research* (PMLR):
Z. Liu, Z. Xu, S. Rajaa, M. Madadi, J. C. S. J. Junior, S. Escalera, A. Pavao, S. Treguer, W.-W. Tu, and I. Guyon. Towards Automated

Deep Learning: Analysis of the AutoDL challenge series 2019. In *NeurIPS 2019 Competition and Demonstration Track*, pages 242–252. PMLR, Aug. 2020c. URL http://proceedings.mlr.press/v123/liu20a.html. ISSN: 2640-3498

— **Paper on theoretical analysis for zero-order meta-learning.** The paper introducing an asymptotic analysis of zero-order meta-learning as a recommendation problem was published in a special issue on meta-learning in *Proceedings of Machine Learning Research* (PMLR):
Z. Liu and I. Guyon. Asymptotic analysis of meta-learning as a recommendation problem. In I. Guyon, J. N. van Rijn, S. Treguer, and J. Vanschoren, editors, *AAAI Workshop on Meta-Learning and MetaDL Challenge*, volume 140 of *Proceedings of Machine Learning Research*, pages 100–114. PMLR, 09 Feb 2021. URL https://proceedings.mlr.press/v140/liu21a.html

— **Paper on DSS & Universal Approximation Theorem.** The paper in which we proved a universal approximation theorem for deep statistical solvers (DSS) was published at Neural Information Processing Systems (NeurIPS 2020). This work is a continuation of the work on LEAP nets introduced in Chapter 6:
B. Donon, Z. Liu, W. LIU, I. Guyon, A. Marot, and M. Schoenauer. Deep Statistical Solvers. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7910–7921. Curran Associates, Inc., 2020c. URL https://proceedings.neurips.cc/paper/2020/file/5a16bce575f3ddce9c819de125ba0029-Paper.pdf

— **Paper on LEAP nets and analysis on super-generalization.** The paper introducing LEAP nets and theoretical analysis of the super-generalization property of LEAP nets was published in *Elsevier Neurocomputing*:
B. Donon, B. Donnot, I. Guyon, Z. Liu, A. Marot, P. Panciatici, and M. Schoenauer. LEAP nets for system identification and application to power systems. *Neurocomputing*, 416:316–327, Nov. 2020b. ISSN 0925-2312. doi: 10.1016/j.neucom.2019.12.135. URL https://www.sciencedirect.com/science/article/pii/S0925231220305051

— **AgEBO paper.** The paper introducing AgEBO was submitted to *The International Conference for High Performance Computing, Networking, Storage, and Analysis* (SC 21) and published as preprint on arxiv.org:
R. Egele, P. Balaprakash, V. Vishwanath, I. Guyon, and Z. Liu. AgEBO-Tabular: Joint Neural Architecture and Hyperparameter Search with Autotuned Data-Parallel Training for Tabular Data. *arXiv:2010.16358 [cs, stat]*, Oct. 2021. URL http://arxiv.org/abs/2010.16358. arXiv: 2010.16358

— **MetaDL challenge design paper.** The paper introducing the design of the MetaDL challenge was published in a special issue on meta-learning in *Proceedings of Machine Learning Research* (PMLR):
A. E. Baz, I. Guyon, Z. Liu, J. N. v. Rijn, S. Treguer, and J. Vanschoren. MetaDL challenge design and baseline results. In *AAAI Workshop on Meta-Learning and MetaDL Challenge*, pages 1–16. PMLR, Aug. 2021. URL https://proceedings.mlr.press/v140/el-baz21a.html. ISSN: 2640-3498

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Apprentissage profond automatisé : principes et pratique

Mots clés: apprentissage automatique automatisé, apprentissage profond, plans d'experiences, sélection de modele

Résumé: L'apprentissage automatique automatisé (AutoML) vise à rendre l'application des méthodes d'apprentissage automatique (ML) aussi dépourvue d'intervention humaine que possible. Cet objectif ambitieux a fait l'objet de nombreuses recherches depuis les débuts du ML. L'objectif de cette thèse est de mettre un cadre formel autour de ce problème aux multiples facettes, de comparer les méthodes existantes et d'explorer de nouvelles directions. Pour formuler le problème AutoML de manière rigoureuse, nous introduisons d'abord un cadre mathématique qui: (1) catégorise tous les algorithmes impliqués en trois niveaux (niveaux alpha, beta et gamma); (2) définit concrètement le concept de tâche (en particulier dans un cadre d'apprentissage supervisé); (3) définit formellement HPO et méta-apprentissage; (4) introduit une métrique d'any-time learning qui permet d'évaluer les algorithmes d'apprentissage non seulement par leur précision, mais également par leur vitesse d'apprentissage. Ce cadre mathématique unifie différents sous-domaines du ML, nous permet de classer systématiquement les méthodes et nous fournit des outils formels pour faciliter les développements théoriques et de futures recherches empiriques. En particulier, il sert de base théorique à une série de challenges que nous avons organisés. En effet, notre principale approche méthodologique pour aborder AutoML avec Deep Learning a été de mettre en place un benchmark étendu, dans le cadre d'une série de challenges sur l'Automated Deep Learning (AutoDL). Ces challenges fournissent une suite de référence de solutions AutoML de base avec un référentiel d'environ 100 datasets, dont plus de la moitié sont publiés sous forme de datasets publics pour permettre la recherche sur le méta-apprentissage. À la fin de ces challenges, nous avons effectué des analyses post-challenge approfondies qui ont révélé que: (1) les solutions gagnantes se généralisent à de nouveaux datasets invisibles, ce qui valide les progrès vers la solution universelle AutoML; (2) Malgré nos efforts pour encourager des solutions génériques, les participants ont adopté des flux de travail spécifiques pour chaque modalité; (3) L'any-time learning a été abordé avec succès, sans sacrifier la performance finale; (4) Bien que certaines solutions se soient améliorées par rapport à la baseline fournie, elles en ont fortement influencé plusieurs; (5) Les solutions d'apprentissage en profondeur dominaient, mais la recherche d'architecture neuronale n'était pas pratique dans les délais impartis; (6) Les études d'ablation ont révélé l'importance du méta-apprentissage, de l'assemblage et du chargement efficace des données, tandis que l'augmentation des données n'est pas critique. Tous les codes et données sont disponibles sur autodl.chalearn.org. Outre l'introduction d'une nouvelle formulation générale du problème AutoML, la mise en place et l'analyse du challenge AutoDL, les contributions de cette thèse comprennent: (1) Développer nos propres solutions aux problèmes que nous avons posés aux participants. Notre travail GramNAS s'attaque au problème de la recherche d'architecture neuronale (NAS) en utilisant une grammaire formelle pour encoder les architectures neuronales. Deux stratégies de recherche alternatives ont été étudiées expérimentalement: une basée sur Monte-Carlo Tree Search (MCTS), qui atteint une précision de 94% sur le dataset CIFAR-10, et une autre basée sur un algorithme évolutif qui bat les packages de pointe AutoGluon et AutoPytorch sur 4 grands datasets bien connus; (2) Former la base d'un futur challenge sur le méta-apprentissage; (3) Apporter plusieurs contributions théoriques. Au cours de cette thèse, plusieurs collaborations ont été engagées pour aborder les problèmes de transfer learning et d'expressivité des réseaux de neurones. Les enquêtes sur le théorème d'approximation universelle nous ont aidés à comprendre la garantie théorique derrière les systèmes d'apprentissage profond que nous déployons.

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title: Automated Deep Learning: Principles and Practice

Keywords: automated machine learning, deep learning, experimental design, model selection

Abstract: Automated Machine Learning (AutoML) aims at rendering the application of machine learning (ML) methods as devoid of human intervention as possible. This ambitious goal has been the object of much research and engineering since the outset of ML. The objective of this thesis is to put a formal framework around this multifaceted problem, to benchmark existing methods, and to explore new directions. To formulate the AutoML problem in a rigorous way, we first introduce a mathematical framework that: (1) categorizes all involved algorithms into three levels (alpha, beta and gamma levels); (2) concretely defines the concept of a task (especially in a supervised learning setting); (3) formally defines HPO and meta-learning; (4) introduces an any-time learning metric that allows to evaluate learning algorithms by not only their accuracy but also their learning speed, which is crucial in settings such as hyperparameter optimization (including neural architecture search) or meta-learning. This mathematical framework unifies different sub-fields of ML (e.g. transfer learning, meta-learning, ensemble learning), allows us to systematically classify methods, and provides us with formal tools to facilitate theoretical developments (e.g. the link to the No Free Lunch theorems) and future empirical research. In particular, it serves as the theoretical basis of a series of challenges that we organized. Indeed, our principal methodological approach to tackle AutoML with Deep Learning has been to set up an extensive benchmark, in the context of a challenge series on Automated Deep Learning (AutoDL), co-organized with ChaLearn, Google, and 4Paradigm. These challenges provide a benchmark suite of baseline AutoML solutions with a repository of around 100 datasets, over half of which are released as public datasets to enable research on meta-learning. At the end of these challenges, we carried out extensive post-challenge analyses which revealed that: (1) Winning solutions generalize to new unseen datasets, which validates progress towards universal AutoML solution; (2) Despite our effort to encourage generic solutions, the participants adopted specific workflows for each modality; (3) Any-time learning was addressed successfully, without sacrificing final performance; (4) Although some solutions improved over the provided baseline, it strongly influenced many; (5) Deep learning solutions dominated, but Neural Architecture Search was impractical within the time budget imposed; (6) Ablation studies revealed the importance of meta-learning, ensembling, and efficient data loading, while data-augmentation is not critical. All code and data are available at autodl.chalearn.org. Besides the introduction of a novel general formulation of the AutoML problem, setting up and analyzing the AutoDL challenge, the contributions of this thesis include: (1) Developing our own solutions to the problems we posed to the participants. Our work GramNAS tackles the neural architecture search (NAS) problem by using a formal grammar to encode neural architectures. Two alternative search strategies have been experimentally investigated: one based on Monte-Carlo Tree Search (MCTS), which achieves 94% accuracy on CIFAR-10 dataset, and another one based on an evolutionary algorithm which beats state-of-the-art packages AutoGluon and AutoPytorch on 4 large well-known datasets; (2) Laying the basis for a future challenge on meta-learning. The AutoDL challenge series revealed the importance of meta-learning but the challenge setting did not evaluate meta-learning properly. With an intern, we experiment with various meta-learning challenge protocols; (3) Making several theoretical contributions. During the course of this thesis, several collaborations were entered to tackle problems of transfer learning and expressiveness of neural networks. Investigations on the Universal Approximation Theorem helped us understand theoretical guarantee behind Deep Learning systems we deploy.